Learning to Complete 3D Scenes from Single Depth Images

Michael David Firman

Thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

UCL Department of Computer Science

Declaration

I, Michael David Firman, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Signed .....

Abstract

Building a complete 3D model of a scene given only a single depth image is underconstrained. To acquire a full volumetric model, one typically needs either multiple views, or a single view together with a library of unambiguous 3D models that will fit the shape of each individual object in the scene. In this thesis, we present alternative methods for inferring the hidden geometry of table-top scenes.

We first introduce two depth-image datasets consisting of multiple scenes, each with a ground truth voxel occupancy grid. We then introduce three methods for predicting voxel occupancy. The first predicts the occupancy of each voxel using a novel feature vector which measures the relationship between the query voxel and surfaces in the scene observed by the depth camera. We use a Random Forest to map each voxel of unknown state to a prediction of occupancy.

We observed that predicting the occupancy of each voxel independently can lead to noisy solutions. We hypothesize that objects of dissimilar semantic classes often share similar 3D shape components, enabling a limited dataset to model the shape of a wide range of objects, and hence estimate their hidden geometry. Demonstrating this hypothesis, we propose an algorithm that can make structured completions of unobserved geometry.

Finally, we propose an alternative framework for understanding the 3D geometry of scenes using the observation that individual objects can appear in multiple different scenes, but in different configurations. We introduce a supervised method to find regions corresponding to the same object across different scenes. We demonstrate that it is possible to then use these groupings of partially observed objects to reconstruct missing geometry.

We then perform a critical review of the approaches we have taken, including an assessment of our metrics and datasets, before proposing extensions and future work.

Acknowledgements

Firstly, I thank my supervisor Simon Julier for his dedicated guidance, patience and knowledge, and for giving me the opportunity and freedom to learn and discover. His attention to detail and knowledge of the subject area have been invaluable. Jan Boehm has also been a tremendous source of information and guidance. I am very grateful for Simon Prince's computer vision lectures, which gave the best foundations in the subject anyone could hope for. In Japan I was fortunate to be able to work under the excellent guidance of Professor Akihiro Sugimoto; どうもありがとうございました[1]. I owe a lot to Gabriel Brostow. He welcomed me in to his Prism group at UCL and has provided a selfless stream of guidance without which much of this work would not have been completed. My examiners, Lourdes Agapito and Walterio W. Mayol-Cuevas, gave up their time to read this thesis and to discuss the content in detail. Their valuable suggestions have improved both this work and, I hope, my future research.

During my PhD I discovered the importance of surrounding myself with a group of very clever people and pestering them on a daily basis for their knowledge. Some of those who have been interrogated the most have been Sara Vicente, who guided me through my first projects; Yotam Doron, for his Python and Git tutorials; Malcolm Reynolds, for his Unix and depth camera expertise; Tom Haines, for all his Blender knowledge, and most of all Oisín Mac Adoha for his contributions to the work on structured shape completion. Others who have gone out of their way to provide knowledge, code and advice over the years include Maciej Gryka, Martin Parsley, Clement Godard, Diego Thomas, Neill Campbell, Yannick Verdie, Peter Rennert, Fabrizio Pece and Daniyar Turmukhambetov. I would not have been able to complete this without your help. I am also extremely thankful for

[1] "Thank you very much for what you have done"

everyone else who has answered questions, read papers and thesis drafts and generally helped make the department an excellent place to do research.

Writing code is hard. I would not have been able to write the algorithms on which this thesis rests without the efforts of the open source community. To anyone who has contributed to an open source project, put their code or data up on their website or answered a StackOverflow question: Thank you.

I would like to give a special mention to some of those outside UCL who have been with me along the way, namely Anne, Pete, Mike, Cat, Maria, Caitríona, Eda, Anna, Toby, Ben, Kat, Marshall, Lee, Isao, Sandy, Yoshi, Aya, Luis, Edmund, Elkie, Jess, Joey, Jamie, Claire, Alex, Tasha, Charlie, Sasha, Cat, Thom, Fiona, Elly, Tom, Claire, Alex, Olly, Joe, Elen, Caz and all the others I have neglected to name here. Without your continual input I might have either finished this a few months earlier or failed entirely. I'm glad I didn't take the risk.

Finally, family. In spite of the tremendous encouragement that friends and colleagues have given, no-one supports my endeavours with such resolute consistency as my grandmother. My sisters have always provided a welcome distraction from work, either through a game, a puzzle or a new scheme being conspired. Last of all, none of this would have been possible without the educational, transportational, nutritional, financial and motivational support my parents have given me. (They also helped roof read this work.)

Contents

Declaration **3**

Abstract **5**

Acknowledgements **7**

1 Introduction **19**

1.1 Application areas 19

1.2 The illusion of occupancy 21

1.3 Challenges 21

1.4 The scope of our work 22

1.5 Our contributions 23

1.6 Thesis outline 24

1.7 Publications 25

2 Problem statement **27**

2.1 Our world model 27

2.2 The projective camera model 29

2.3 The inverse problem of occupancy recovery 30

2.4 How to evaluate occupancy prediction 30

3 Background **35**

3.1 Specialist hardware for scene completion 35

3.2 Generative models of shape and appearance 36

3.3 Fitting full 3D models 38

3.4 Completion using the observed scene . . . 39
3.5 Using 3D primitives for recognition and completion . . . 40
3.6 Surface completion . . . 41
3.7 Summary . . . 43

4 Learning per-voxel occupancy for scene completion 45
4.1 Problem statement . . . 46
4.2 Background . . . 46
4.3 A synthetic dataset for occupancy prediction . . . 54
4.4 A tabletop dataset for occupancy prediction . . . 56
4.5 Our method for voxel occupancy prediction . . . 57
4.6 Evaluation . . . 66
4.7 Conclusions . . . 78

5 Structured completions of 3D scenes 81
5.1 Related work . . . 83
5.2 Overview of our structured prediction algorithm . . . 85
5.3 Learning a mapping from features to voxlets . . . 87
5.4 Predicting occupancy at test time . . . 90
5.5 Evaluation . . . 96
5.6 Conclusions and future work . . . 108

6 Learning to discover objects for object completion 111
6.1 Problem statement . . . 113
6.2 Related work . . . 113
6.3 Our object discovery methodology . . . 119
6.4 Using discovered objects to complete scenes . . . 126
6.5 Evaluation . . . 128
6.6 Conclusions . . . 143

7 Conclusions 151
7.1 Our occupancy prediction methods . . . 151
7.2 Analysis of evaluation metrics . . . 155

7.3 Is the ground truth good enough? . . . 157
7.4 Future work for solving the occupancy problem . . . 158
7.5 Future work extending the scope of the problem . . . 160

Appendices **163**

A Notation **165**

B All images from the tabletop dataset **167**
B.1 Fold 1 (training) . . . 167
B.2 Fold 2 (training) . . . 168
B.3 Fold 3 (test) . . . 169

C RGBD features **171**
C.1 Bounding box size . . . 171
C.2 RGB mean . . . 172
C.3 RGB histogram . . . 172
C.4 Shape distributions . . . 173
C.5 Histogram of Gradients (HOG) . . . 173
C.6 Spin images . . . 174

D The adjusted Rand index **177**

Bibliography **179**

List of Figures

1.1 An overview of our occupancy model of the world 20

1.2 Some of the challenges associated with understanding the world. 22

2.1 The truncated signed distance function world model 28

2.2 An illustration of the ambiguous nature of our inverse problem 30

2.3 The region used for evaluation of our algorithms. 31

2.4 Examples of high and low precision and recall 32

2.5 Potential problems with the IoU metric 33

3.1 Examples of geons used for describing objects 41

4.1 An overview of the voxel filling method of Zheng et al. 47

4.2 Different datasets are captured in different styles 52

4.3 A reconstruction from the SUN3D dataset 53

4.4 The process for creating and rendering synthetic scenes. 55

4.5 Screenshots of the synthetic scene generation process 56

4.6 Examples from our tabletop dataset 57

4.7 Training objects from the tabletop dataset 58

4.8 Examples from our synthetic dataset 58

4.9 Labelling of a voxel grid according to the camera observations 59

4.10 An illustration of a hypothesis about voxel occupancy 59

4.11 The Axis-aligned Voxel Occupancy Feature 61

4.12 The 26 directions for the AVOF 62

4.13 Limitations of the AVOF feature representation 63

4.14 How the surface feature is used to make predictions 64

4.15 How a single surface feature is computed from a query point 65
4.16 The pipeline used for segmentation for the baseline implementation. 67
4.17 Qualitative results from the implicit predictions on the synthetic dataset. . 70
4.18 A comparison against baselines on a scene from the synthetic dataset 72
4.19 Performance of the model with the training voxels 73
4.20 Qualitative results from the implicit predictions on the tabletop dataset. . . 74
4.21 Sensitivity of the completion to changes in camera viewpoint 75
4.22 The different angles of elevation in the synthetic dataset 76

5.1 Examples of the *shape sharing* algorithm for image segmentation 82
5.2 An example completion using our structured prediction algorithm 83
5.3 A 2D overview of the structured prediction algorithm 86
5.4 How the classification loss is used to make splits at nodes in the forest . . . 89
5.5 Predicted voxlets shown in context 91
5.6 Pre-segmentation and sample locations for a single image 91
5.7 The pipeline for selecting from the tree predictions for a single query point 92
5.8 Grounded and floating voxlets motivation 95
5.9 The voxlet sizes and coordinate systems used in most of our experiments . 96
5.10 Example results from voxlet completion on the BigBIRD dataset 98
5.11 Qualitative comparison of voxlet selection strategies for our tabletop dataset 99
5.12 Example completions using voxlets on the synthetic scene dataset 100
5.13 Occupancy predictions for two scenes from the NYU-Depth V2 dataset . . . 101
5.14 The effect of varying the voxlet size on the prediction performance. 104
5.15 A distribution over the most popular voxlets in the testing dataset. 105
5.16 Some of the most and least popular voxlets 105
5.17 Sensitivity of the completion to changes in camera viewpoint 106
5.18 Proposed future work regularising the voxlet predictions 110

6.1 Motivation image for object discovery 112
6.2 Using object discovery to complete missing geometry 114
6.3 Object cosegmentation from Vicente et al. 116
6.4 Example results from Shin et al. 117

6.5 Looking at different properties can give very different clusterings. 117
6.6 Inputs and outputs of a supervised clustering algorithm. 119
6.7 Pictorial overview of our segmentation pipeline 120
6.8 Overview of the weight assignment algorithm 121
6.9 Examples from the RGB-D dataset 129
6.10 ROC curves for pairwise classification accuracy 130
6.11 Qualitative pairwise results from the RGBD turntable dataset 131
6.12 Qualitative clustering results from the RGBD object dataset 133
6.13 Confusion matrix for the object discovery performed on dataset $\mathfrak{D}_1$ 134
6.14 Comparison of our correlation clustering implementation 135
6.15 Some qualitative segmentation results 136
6.16 Examples of different unary scores 138
6.17 Receiver-operator characteristic curve for unary scores 139
6.18 Pairwise results examples 139
6.19 Receiver-operator characteristic curve for pairwise scores 140
6.20 Some objects found shown in the context of the original images. 140
6.21 Clusters found from our real-world dataset 146
6.22 Confusion matrix for the object discovery performed on our scene dataset . 147
6.23 Quantitative comparison of the correlation clustering algorithm 147
6.24 Qualitative object discovery results on the tabletop dataset 148
6.25 An illustration of how 'pre-clustering' could reduce algorithm complexity . . 149

7.1 A comparison of algorithms image 00207_[536] 154
7.2 A comparison of algorithms on image 00218_[121] 155
7.3 Proposed Mechanical Turk system to evaluate completions 156
7.4 Examples of missing data in the ground truth volumes 157
7.5 New datasets which could be modified to allow for use with our algorithms 159
7.6 A concept for dense voxel semantic labelling 161

C.1 Illustration of spin image creation 176

D.1 Examples of ARI scores for different levels of inaccuracy 178

List of Tables

4.1 A review of publicly available datasets captured with a 3D sensing device . 50

4.2 Statistics of the tabletop dataset 57

4.3 Implicit prediction results on the synthetic dataset 69

4.4 Implicit prediction results on the tabletop dataset 73

4.5 How the angle of elevation affects results on the synthetic dataset 76

4.6 Per-object results with the per-voxel algorithm 77

5.1 Evaluation of different completion methods on the BigBIRD dataset 97

5.2 A quantitative evaluation of the voxlets algorithm on the tabletop dataset . 97

5.3 A quantitative evaluation of the voxlets algorithm against our baseline . . . 99

5.4 Quantitative evaluation of oracle-enhanced versions of the voxlets algorithm 103

5.5 Per-object results with the voxlets algorithm 107

6.1 Algorithms used for object discovery 116

6.2 The features used in our object discovery algorithm. 121

6.3 The train/test split of the turntable dataset 129

6.4 Object discovery completion evaluated on our tabletop dataset 143

7.1 Quantitative results across all algorithms 153

7.2 Strengths and weaknesses of algorithms in this thesis 153

A.1 Types of symbol used in this thesis 165

D.1 The four sets used for calculation of the ARI 177

Chapter 1

Introduction

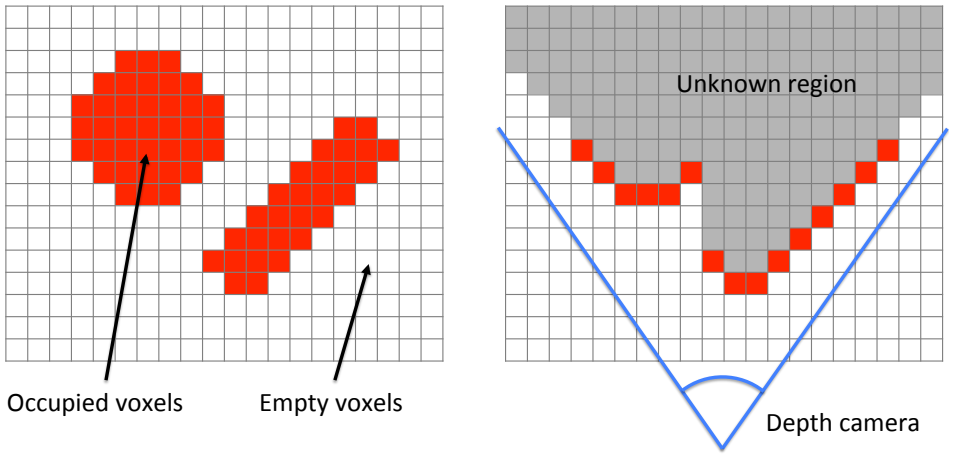
It is critical to have a complete representation of the world's geometry for many applications. When a robot hand or autonomous vehicle interacts with an unknown object in an unknown environment, a full 3D understanding can greatly help to navigate and prevent collisions. In photo-editing, full geometry would enable realistic shadows from a new light source to be automatically added to an image or stereo pair after capture.

We can broadly categorize space in our world as being 'occupied' and opaque, or 'empty' and transparent. Depth cameras such as the Microsoft Kinect are able to give an estimate of which regions of a scene are composed of free, empty space. However, each pixel in a depth image only makes an estimate of occupancy in front of the first solid surface encountered along that camera ray (Figure 1.1). Occlusion prevents any information from being measured about the occupancy of space beyond that first surface.

A large amount of computer vision research has been devoted to reconstructing a full 3D world model from RGB or depth images of a scene captured from multiple viewpoints, thus coping with the effects of occlusion (e.g. [145, 80, 103, 173]). We instead focus on the task of classifying each voxel in a local 3D scene as being either 'occupied' or 'empty', given just a single depth image from one viewpoint.

1.1 Application areas

There are many application scenarios in which capturing enough viewpoints for a multi-view 3D reconstruction is difficult or impossible. For example:



(a) Voxel world model (b) Depth rendering of world

Figure 1.1: We model the space in our world as being either 'occupied' or 'empty'. An overhead view of a coarse 2D representation of this is shown in (a), with red regions as 'occupied' and white regions as 'empty', according to ground truth. When observed by a depth camera, only the first occupied location along each ray is seen. This leaves a region of unknown occupancy extending beyond the depth surface, shown as grey in (b). The aim of our algorithm is to predict the occupancy state of this unknown region.

⇛ A wheeled robot entering a doorway of a cluttered room it has never seen before may wish to navigate to a target object at the far side. However, the floor surface may be occluded for much of the route, by furniture and other clutter. Standard robot path planning approaches typically assume occluded regions of the scene are 'unknown', giving them a fixed prior occupancy probability until more sensor data arrives [43, 158]. By making a prediction of the full 3D shape of the scene the robot will be able to use this as a more informed prior for a plan of a route to the target object, using methods such as [47, 76].

⇛ If a single photo is captured from a mobile device with a depth camera attached, an appealing post-processing option is to be able to relight the scene by digitally adding light sources. To cast realistic shadows, however, 3D geometry is required [109, 90, 163].

⇛ In multi-view reconstruction problems it is often useful to have an initial estimate of the shape of the scene from the first few frames, which can be replaced as more footage comes in. This estimation can be used as a prior to solve the next-best-view problem, deciding where the camera should be moved to capture the most informative next image of the scene [83]. We can also consider that if parts of the

environment can be well-predicted, we can avoid visiting some parts of the scene altogether.

1.2 The illusion of occupancy

A precise definition of an occupied region in 3D space is hard to define. We could define a portion of space as being occupied if it contains solid matter, and vacant if it contains gas or a vacuum[1]. While this is a very natural definition of occupancy, it is of limited practical use.

When we, or a robot, interact with the world, we care about the external surfaces rather than the internal geometry. Consider the example of a closed cardboard box. Visual inspection by a camera or human observer provides no clues about the solidity of the object beyond the outer surface. However, inside the box there could be any possible arrangement of solid or vacant regions. While gaining knowledge of this is impossible without opening the box, such a detailed representation is not required for a a robot interacting with the scene, which cares primarily about the external surfaces.

We instead re-pose our problem. We state that we aim to predict *whether or not each location in the scene could be viewed as being empty, if a camera were to move all the way around the scene from a given starting location.* In effect, we strive to predict the voxelized output of KinectFusion [80], but using only a single depth image of the scene as input instead of multiple views. Under this simplification, our work is still suitable for use in the application areas described above (e.g. robot navigation), but we greatly simplify the method of collecting training and evaluation data.

1.3 Challenges

This task has many inherent challenges:

- ⇛ There are an exceptionally large number of latent properties of objects and scenes in our world. Objects can appear in a large number of different arrangements, with large variations in shape, colour and lighting making it very difficult to form models over shape (Figure 1.2).

[1] For the purposes of this thesis we only consider solids and free space, to limit the scope.

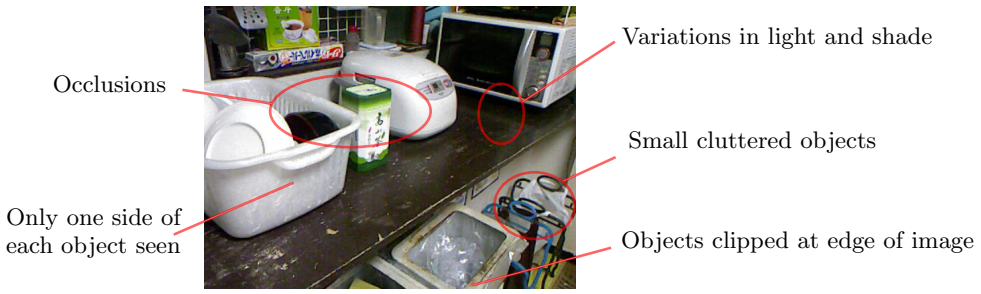


Figure 1.2: Some of the challenges associated with understanding the world.

⇛ Data acquired from sensors can be of very low quality, and subject to effects such as sensor noise, limited pixel resolution and the quantisation of depth and colour measurements into discrete 'bins'. This is especially true of the low-quality consumer sensors which we use in our work.

⇛ There are many potential solutions, given one set of input data. We hence rely on statistical models of the world in order to make our predictions.

⇛ Data collected from the three-dimensional world tends to be very large. A Kinect scanner, for example, is capable of collecting around 37MB of data every second. This can make it difficult to develop algorithms to process 3D data.

⇛ Conversely, there are relatively few datasets which are suitable for learning or evaluating occupancy prediction systems. There are fewer datasets of depth images than there are of RGB data, and datasets that do exist tend not to capture the full geometry of scenes, instead only capturing single images or short videos.

1.4 The scope of our work

The problem as currently described is poorly bounded. We therefore make some limitations on the scope of the system:

⇛ We assume we have as input to our algorithm a single RGBD image captured of a scene from a 3D scanning device with known intrinsic properties.

⇛ We restrict ourselves to static indoor scenes containing man-made objects. This lends itself to applications of robotic interaction, and allows us to use existing depth

datasets [49]. Work performed here on indoor scenes could, with modification, be applied to data collected outdoors, for example from lidar scanners.

⇛ For all our algorithms presented in this thesis, we make use of supervised learning. This means that we make the assumption that we are able to get training data which in some way is representative of the data we expect to test on. For different aspects of our work we assume different levels of additional data and supervision has been provided.

⇛ We assume we are able to get access to ground truth occupancy data for training. We discover in our work that gaining such ground truth data can be challenging in spite of recent innovations in 3D reconstruction work; we discuss these difficulties in Section 7.3.

1.5 Our contributions

We make five primary contributions in this thesis:

Contribution 1: Voxel occupancy prediction datasets: We perform a review of existing RGBD datasets, and use this to motivate the need for a new dataset to train and evaluate voxel occupancy prediction. We introduce a synthetic dataset and a tabletop dataset which we can use for this purpose (Chapter 4)[2].

Contribution 2: Learning to predict the occupancy of each voxel: Using two different novel feature representations, we learn to predict whether or not each unobserved voxel in a scene is occupied or not (Chapter 4). We demonstrate the qualities of this *per-voxel* framework on both of our novel datasets.

Contribution 3: Structured prediction of voxel occupancy: In Chapter 5, we use a structured prediction algorithm to predict voxel occupancy. Using a novel framework we are able to complete shape with more regularity and smoothness than our single-voxel predictions, using training data to learn a mapping from a depth image to a full 3D grid's occupancy.

[2]Datasets available from `http://visual.cs.ucl.ac.uk/pubs/depthPrediction/`

Contribution 4: Object discovery from RGBD images using supervised clustering: We present a novel method for *discovering* recurring instances of objects in a set of RGBD images (Chapter 6). Using a novel probabilistic framework incorporating a supervised clustering algorithm, we show that a small amount of hand-labelled training data can be used to discover these repeating object instances. This technique removes dependency on the user-specified parameters which most unsupervised methods rely on.

Contribution 4: Completing 3D scenes by matching completed objects in other scenes: We demonstrate how we can use discovered object matches (Contribution 4) can be used to completing regions of 3D scenes, under the assumption that at least one scene containing each object has had its full geometry captured. This concept is presented in Chapter 6.

1.6 Thesis outline

This thesis is structured as follows:

- ⇛ In Chapter 2 we formally set out our mathematical model of the world and define the problem we wish to solve.
- ⇛ In Chapter 3 we discuss previous approaches to solve our problem of scene occupancy. In this chapter we justify why our novel approaches are required.
- ⇛ In Chapter 4 we present an alternative method to solve the occupancy problem. Here we introduce a novel feature descriptor which we use to predict the occupancy of each unknown voxel in a scene. In this chapter we also introduce a review of RGBD datasets, and use this to motivate the introduction of two novel datasets with which we evaluate our algorithm.
- ⇛ In Chapter 5 we present a novel framework for making structured predictions of occupancy, again using a set of training scenes to learn the mapping from depth image to occupancy prediction.
- ⇛ In Chapter 6 we introduce a novel method to understand images of RGBD scenes by finding correspondences between regions in different images which represent the

same object. We then show that these discovered corresponding regions can be fused together to complete missing regions of geometry.

⇛ Finally, in Chapter 7 we evaluate the strengths and weaknesses of our different approaches to the problem. We assess the scenarios in which each approach could be beneficial and we discuss the opportunities for improvements beyond the work presented in this thesis.

1.7 Publications

Some of the work in this thesis has been published previously. The work on object discovery presented in Chapter 6 was published as:

⇛ Michael Firman, Diego Thomas, Simon Julier and Akihiro Sugimoto. *Learning to Discover Objects in RGB-D Images Using Correlation Clustering.* In proceedings of the International Conference on Intelligent Robots and Systems (IROS), 2013.

The ideas in Chapter 6 were partly influenced by the findings of the following publication. This was the first analysis of the influence of noise on the input features for topic models and their use for clustering multiple views of objects together:

⇛ Michael Firman and Simon Julier. *'Misspelled' Visual Words in Unsupervised Range Data Classification: The Effect of Noise on Classification Performance.* In proceedings of the International Conference on Intelligent Robots and Systems (IROS), 2011.

The work on structured prediction of voxel occupancy (Chapter 5) has been accepted for publication as:

⇛ Michael Firman, Oisin Mac Aodha, Simon Julier and Gabriel Brostow. *Structured Prediction of Unobserved Voxels From a Single Depth Image.* In proceedings of Computer Vision and Pattern Recognition (CVPR), 2016 (to appear).

The review of RGBD datasets in Chapter 4 was extended to include datasets of humans and faces, and published as:

⇛ Michael Firman. *RGBD Datasets: Past, Present and Future.* In proceedings of CVPR Workshop on Large Scale 3D Data: Acquisition, Modelling and Analysis, 2016 (to appear).

Chapter 2

Problem statement

Before trying to solve our overall problem we first formalise it mathematically. This helps to rationalise the steps we take to solving our problem and introduces mathematical notation.

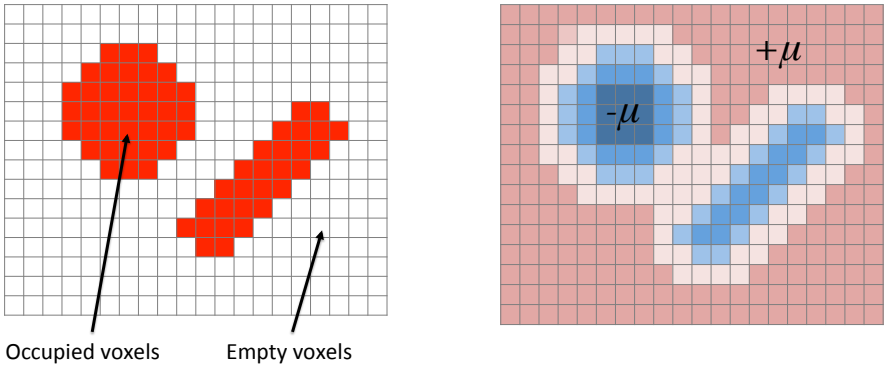
2.1 Our world model

We can model the world as a regular 3D grid of voxels $\mathbf{O}$, where $o_{i,j,k}$ is the voxel at the i^{th} of I rows, the j^{th} of J columns and the k^{th} slice of K slices. Each $o_{i,j,k} \in \{0, 1\}$ is a binary variable which takes the value 0 if the voxel is vacant, and 1 if it is occupied, according to our definition of occupancy in Section 1.2.

However, we are able to encode additional information at each voxel by making use of a *truncated signed distance function* (TSDF), following works such as [33, 80, 131]. In the TSDF model, we maintain a similar grid of voxels $\mathbf{V}$. In comparison though, each voxel $v \in \mathbf{V}$ now gives its measure of occupancy as a real number $v \in [-d_{\max}, d_{\max}]$. Each $|v_{i,j,k}|$ gives the Euclidean distance in world space from (i, j, k) to the nearest surface (zero-crossing) in $\mathbf{V}$, truncated to a maximum value of $d_{\max}$, which is a parameter. v is negative if voxel (i, j, k) is inside solid opaque matter, and positive if it is in free space. The zero level-set of $\mathbf{V}$ therefore implicitly represents the surface.

From the TSDF voxel grid $\mathbf{V}$ we can recover our original occupancy grid $\mathbf{O}$ using the relation

$$\mathbf{O} = (\text{sgn}(\mathbf{V}) + 1)/2. \tag{2.1}$$



(a) Voxel world model (b) TSDF world model

Figure 2.1: We model our world as a grid of voxels, each of which is either occupied or empty (vacant). An overhead view of a coarse 2D representation of this is shown in (a). In our algorithms, we use a truncated signed distance function to represent the distance from each voxel to the nearest surface. This is shown in (b), where blue voxels represent negative values and red voxels positive values.

Using a signed distance function offers advantages over a binary world model:

⇛ A more fine-grained surface is able to be encoded in the same resolution grid as the embedded zero level-set can be recovered at sub-voxel accuracy.

⇛ Each voxel location is able to encode non-local information about the world as it can suggest the precence or absence of a surface many voxels away from its location.

Using a *truncated* signed distance field, as opposed to an *unbounded* signed distance field, offers the following advantages:

⇛ A smaller amount of memory is required at each location. [80] store each voxel occupancy as an 8-bit signed integer, truncated at $[-127, 128]$.

⇛ At test time we find that we may want to combine together multiple predictions for a single voxel. When these predictions are truncated values it adds a level of robustness; in effect, each candidate prediction can only be 'wrong' by a maximum value of $2d_{\max}$.

We give a stylised illustration of the TSDF in Figure 2.1.

2.1.1 Indexing notation for 2D and 3D arrays

For ease of notation we define

$$\mathbf{V}(\mathbf{g}) \equiv v_{i,j,k}, \quad \text{where} \tag{2.2}$$

$$\mathbf{g} = (i, j, k) \in \mathbb{Z}^3. \tag{2.3}$$

This allows us to easily refer to the value of the voxel at coordinate position (i, j, k) with the notation $\mathbf{V}(\mathbf{g})$, similar to indexing conventions in languages such as MATLAB and NumPy. We follow an analogous convention for indexing in 2D matrices.

2.2 The projective camera model

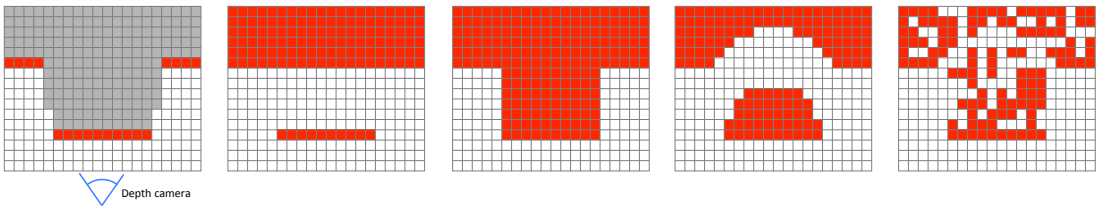
We assume a world coordinate system centred at $i = j = k = 0$, aligned with the principle grid directions. Each voxel is of real-world dimensions $\eta \times \eta \times \eta$, meaning that each voxel has its centre at position $(\eta i, \eta j, \eta k)$.

Each scene, represented as a voxel grid, is then captured in a single static depth image $\mathbf{D}$ by a camera at world position $\mathbf{t} \in \mathbb{R}^3$, with local orthogonal coordinate system $\mathbf{R} \in \mathbb{SO}_3$, focal length f and camera centre (c_x, c_y). We model the capture process as a projective system [160]. Under this model, a voxel at index position (i, j, k) in $\mathbf{V}$ projects into the camera to a pixel location of (α, β) at a depth of d using the relationship

$$d \begin{bmatrix} \alpha \\ \beta \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} [\mathbf{R}, -\mathbf{Rt}] \begin{bmatrix} \eta i \\ \eta j \\ \eta k \\ 1 \end{bmatrix}. \tag{2.4}$$

Here d is the perpendicular distance from the camera centre to the voxel. Finally, we model the value of a pixel at location (α^*, β^*) in the depth image $\mathbf{D}$ as the depth to the first occupied voxel along the camera ray:

$$\mathbf{D}_{\alpha^*,\beta^*} = \min_{i,j,k} \{d_{i,j,k} : v_{i,j,k} < 0, \lfloor \alpha_{i,j,k} \rfloor = \alpha^*, \lfloor \beta_{i,j,k} \rfloor = \beta^*\}. \tag{2.5}$$



(a) Input view (b) (c) (d) (e)

Figure 2.2: An illustration of the ambiguous nature of our inverse problem: In (a) we see the view of an example scene captured by a depth camera. (b), (c), (d) and (e) show possible completions. In this example there are over 1.6×10^{60} scenes which would give the input view as shown in (a).

2.2.1 Limitations of our projective camera model

Our projection model is deterministic. This is a reasonable reflection of reality; images captured by the same sensor at the same position in the same scene under the same conditions tend to be highly similar. However, there is a stochastic element to the true image capture process due to sensor noise, effects of lighting and so forth. In our model we also do not model noise, nor do we explicitly account for the quantisation effect present in many sensing devices. For example in the Kinect sensor depth measurements are normalised and quantised into a fixed number of 'bins'. These effects are more apparent in images captured from our depth sensors than under a traditional two-dimensional camera model.

2.3 The inverse problem of occupancy recovery

The aim of the work presented in this thesis is to recover $\mathbf{V}$ given $\mathbf{D}$. This is an ambiguous inverse problem. The $\min_{i,j,k}$ in Equation 2.5 is the manifestation of occlusion: each ray from the camera stops at the first occupied voxel it reaches. This occlusion means that the data presented, $\mathbf{D}$, is insufficient to uniquely recover $\mathbf{V}$. In fact there are typically an intractably large number of completions of $\mathbf{V}$ which could explain the input view (Figure 2.2). We therefore rely on forming models of the world in order to approximate a solution.

2.4 How to evaluate occupancy prediction

Given a ground truth occupancy grid $\mathbf{O}_{\mathtt{gt}}$ and a predicted occupancy grid $\mathbf{O}_{\mathtt{pred}}$, we desire a mapping to a scalar evaluation metric, such that predictions which are in some

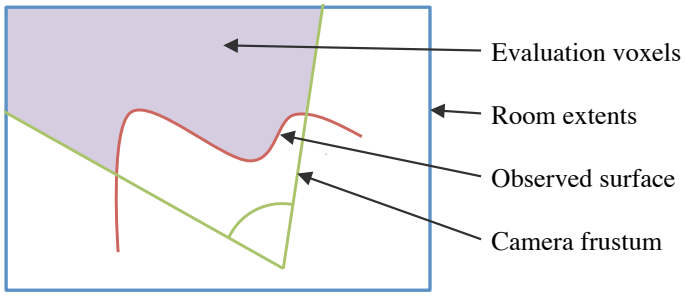


Figure 2.3: The region used for evaluation of our algorithms.

way 'better' achieve higher scores. One simple measure of accuracy would be to find the average number of voxels which have been correctly classified as being 'occupied' or 'vacant'. However, this measure would be highly skewed by class imbalances in the two classes. There are likely to be far more vacant than occupied voxels, so a high accuracy score could be achieved by setting all unknown voxels to 'vacant'. Instead, we take inspiration from works on detection and semantic segmentation and report scores of precision, recall and intersection over union [32].

We make sure to only evaluate voxels which are in the camera frustrum, i.e. all the voxels in the set $\mathcal{F}$. The set of visible voxels are defined as

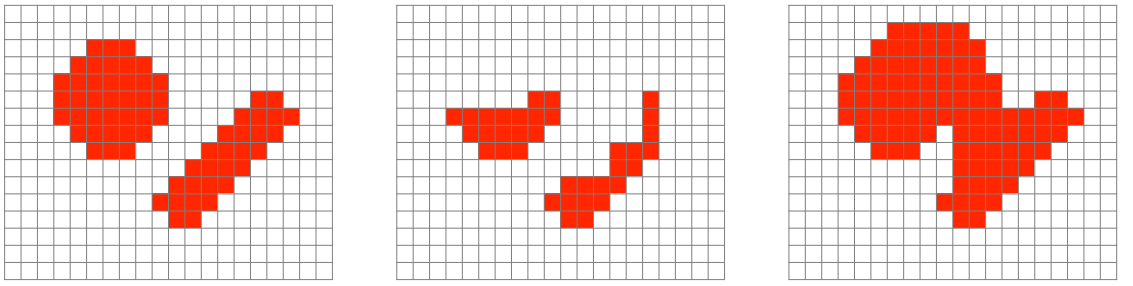
$$\mathcal{F} = \{(i, j, k) \quad : \quad 0 \geq \alpha < w, 0 \geq \beta < h, \}, \tag{2.6}$$

where (w, h) are the width and height of the output image and (i, j, k) are computed using Equation 2.5. In addition, we ensure we only evaluate on voxels behind the depth image as viewed by the camera. Where scenes have natural boundaries, such as walls or other constraints, we limit the evaluation region to voxels falling within these boundaries. This evaluation region is depicted in Figure 2.3

Our three metrics are then computed as follows.

Precision Out of all the voxels which are occupied in the predicted grid, the precision measures the fraction which are also occupied in the ground truth:

$$\texttt{Precision} = \frac{\sum_{\mathbf{f} \in \mathcal{F}} [\mathbf{O}_{\texttt{gt}}(\mathbf{f}) \wedge \mathbf{O}_{\texttt{pred}}(\mathbf{f})]}{\sum_{\mathbf{f} \in \mathcal{F}} [\mathbf{O}_{\texttt{pred}}(\mathbf{f})]} \in [0, 1] \tag{2.7}$$



(a) Ground truth

(b) A prediction with high precision but low recall

(c) A prediction with high recall but low precision

Figure 2.4: Examples of high and low precision and recall

Recall Out of all the voxels which are occupied in the ground truth grid, the recall measures the fraction which are also occupied in the prediction:

$$\texttt{Recall} = \frac{\sum_{\mathbf{f}\in\mathcal{F}}[\mathbf{O}_{\texttt{gt}}(\mathbf{f}) \wedge \mathbf{O}_{\texttt{pred}}(\mathbf{f})]}{\sum_{\mathbf{f}\in\mathcal{F}}[\mathbf{O}_{\texttt{gt}}(\mathbf{f})]} \in [0,1] \tag{2.8}$$

A perfect recall could be achieved simply by setting all the predicted voxels to 'occupied'. We give examples of predictions with high and low precision and recall in Figure 2.4.

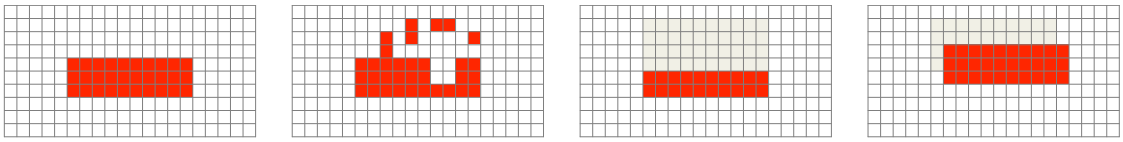
Intersection over union (IoU) The intersection over union is a fraction of the number of voxels which are either occupied in the ground truth *or* the prediction, which are occupied in both.

$$\texttt{IoU} = \frac{\sum_{\mathbf{f}\in\mathcal{F}}[\mathbf{O}_{\texttt{gt}}(\mathbf{f}) \wedge \mathbf{O}_{\texttt{pred}}(\mathbf{f})]}{\sum_{\mathbf{f}\in\mathcal{F}}[\mathbf{O}_{\texttt{gt}}(\mathbf{f}) \vee \mathbf{O}_{\texttt{pred}}(\mathbf{f})]} \in [0,1] \tag{2.9}$$

The IoU is commonly used as a metric for object detection, e.g. [135]. It can be seen as a balance between precision and recall, and hence we use it as the main evaluation criterion.

2.4.1 Limitations of evaluation metrics

However, these evaluation metrics have their shortcomings. In particular, they do not especially favour smooth and contiguous completions. This means that noisy predictions can end up with a higher score than smooth, plausible completions which under- or over-estimate the size slightly. An example of this phenomenon is shown in Figure 2.5. In Section 7.2 we evaluate further qualities of the metrics. We also suggest some alternative application-specific measures of success and failure. Three alternative evaluations we



(a) Ground truth

(b) A noisy prediction, with IoU = 0.70

(c) A smooth and plausible prediction, with IoU = 0.67

(d) A good completion but offset by one voxel. IoU = 0.45

Figure 2.5: Examples of how a noisy, implausible completion can gain a higher IoU score than slightly mis-sized or mis-aligned completions.

propose in Section 7.2 are:

- A metric based on the ability for a robot to use the completion to successfully navigate to an occluded region of a scene
- A grasping metric, examining if the predicted completion enables a robot to successfully grasp an object
- A perceptual evaluation of the quality of completions using humans to evaluate completions

Chapter 3

Background

Previous works on completing unknown regions of visual data can be categorized against several different criteria.

Works can be classed according to their application domain, such as operating on meshes [68, 144], 2D images [64] or, like ours, in voxel space [91]. Some works make use of highly specialist hardware to capture images e.g. [167], while we use hardware available to consumers. Some approaches only aim to get a result that looks plausible to a human, while we strive for accuracy in comparison to the ground truth. In comparison to works which use heuristics, e.g. [178], we make use of training data. Such supervised approaches can be further classed according to whether the data used come from within the same image (e.g. symmetry [99]) or from a database of training data, as in our work.

3.1 Specialist hardware for scene completion

Our stated problem of scene completion corresponds roughly to the long sought-after sci-fi scenario of *X-ray vision* or the ability to *see around corners*. While X-ray scanners can build up three-dimensional images, they rely on hardware being present both in front of and behind the objects being scanned, violating our single viewpoint requirement. Velten et al. [167] present a method to 'see around corners' using an ultra-fast time of flight camera to combine diffuse reflections from the rear of objects off of surrounding surfaces. This requires the rear of objects to be seen in reflection off of other objects in the scene. Their results are ground-breaking in their novelty, yet they show poor levels of detail and

their experimental setup is extremely limited.

In our work we make use of safe, commodity hardware to make our predictions of the scene from one viewpoint.

3.2 Generative models of shape and appearance

One possible method of completion from a single depth image captured from simple hardware is to use a *generative model* over shape. In one form of generative model, a joint distribution is formed over observed and latent variables. This means that given a novel scene where some of the voxel states are unknown, these could be inferred using the visible variables and a learned model over latent variables. Alternatively, a *prior* over plausible completions could be combined with a *data term* measuring the 'goodness-of-fit' to the observed data. Bayes Rule can then be used to find a posterior distribution over completions which both match the prior and the observed data.

Generative models over images and scenes have been used to good effect for tasks such as segmentation and labelling [164]. One problem with using learned generative models for *completion* is that it is difficult to capture fine-grained details using generative models, which typically operate using a *subspace model* using parametric probability distributions e.g. Mixtures of Gaussians. Fine-grained details were able to be added to such a model by Mohammed et al. [115], who performed inpainting (and generation) of images of faces by adding high-resolution patches on top of a low-resolution sample from a generative model.However their algorithm relies on their training and test-time images of faces being co-aligned in image space, such that, for example, the mouth is always in the same place in each image. Subspace models have been extended to allow for objects with varying locations [28] and numbers [165] of parts. These extensions have been used to good effect in the completion of RGB images [165], typically being used to model the appearance of isolated or segmented objects.

In recent years, some forms of deep learning have become popular additions to the generative machine learning toolset. Eslami et al. [45] use a modified version of a deep Boltzmann machine to form a generative model over binary images, where the model consists of learned pairwise interaction weights between hidden and visible units. This

allows them to perform inpainting on these images. However, their images are very small (32×32 pixels) and are all aligned in approximately the same way (e.g. a set of horses all facing the same direction). Furthermore, they only train one model per object class. The most important restriction for our purpose is that they are only able to learn a model for a fixed size image, while in our case each scene may be of a different size.

Wu et al. [175] extended this concept into 3D, using a variant of convolutional deep belief networks to complete voxel grids from depth images. However, they do suffer from limitations: They only learn a model for single objects, not objects in occlusions and clutter, and their occupancy predictions are restricted to fixed sized grids, like the work of Eslami et al. [45]. These restrictions make it difficult to learn from real scenes; instead they limit themselves to learning shape from a large CAD model dataset.

Three fundamental problems with directly using generative approaches for scene completion are:

The prior Where a prior is defined for use in a generative model, it can be difficult to accurately capture the subtleties of the shapes and object arrangements present in the world.

Dimensionality A learned generative model for completion is often encoded as pairwise measures between observed variables. For 2D images, the storage and use of such pairwise measures (and the modelling of their covariance) is tractable. However, in 3D the dimensionality means that such a naive representation can quickly become intractable.

Inference-time tractability Discriminiative models typically lend themselves to fairly fast evaluation times. Inference with generative models often takes the form of sampling over the output space, which can make them slow to evaluate.

Output grid size Generative models which learn distributions over pixels or voxels typically operate on fixed-size grids (e.g. [175, 45]). In our case, we assume we do not know the size of the output grid until test time, making it difficult to exploit these types of generative approaches.

In our work, we use a discriminative approach for speed of tractability and to be able to

exploit the predictive power of discriminative machine learning algorithms.

3.3 Fitting full 3D models

Forming a generative model over all possible configurations of occupied voxels is difficult, due to the difficulties involved in making the inference tractable and the fixed size output typically associated with generative representations. A way to avoid having to learn a full model over shape is to use prior knowledge about the type and shape of objects present in the scene. If 3D models are available for objects known to be in the scene, then these 3D models can be fitted to a depth image using a 6 degree-of-freedom transformation. The ground truth occupancy of each 3D model, together with the inferred transformation, can then be used to infer the occupancy of the full scene.

Fitting instance-specific 3D models In many cases, this problem of model fitting is framed as assuming that an exact, instance-specific training instance is available. These find object pose in many different ways, for example Drost and Ilic [41] aggregate pairwise features in a hash table, while Hinterstoisser et al. [73, 74] use a combination of gradients of edges and texture. These methods rely on having a training model of every item that could possibly be seen, rendered from multiple viewpoints. This is infeasible for most practical situations, as such training data is expensive to produce, subject to human bias and most importantly limits the number of objects that can be recognized to those present in the training set. For example, while the impressive RGB-D Object Dataset [101] contains views of 300 objects, there are 9,500 distinct items in the current IKEA product range alone [79].

Fitting generic, class-level 3D models All the instance-level model fitting methods assume that the item in the scene has the exact same geometry of the training instances. Some works focus on the broader problem where an exact match is not present in the training set. Shen et al. [147] complete the missing regions of a single object using an assembly of parts from several different models in a CAD database. Cocias et al. [27] take an alternative approach, where they have a single primitive object for each class, which they fit to the point cloud and allow to deform to account for inter-class variation.

Prisacariu and Reid [131] present a more developed version of this idea where the latent axes of variation of the shape of the object being fitted are allowed to deform during the pose estimation. The concept of adaptive fitting of generic primitives to the specific instances seen in an image has seen practical application in the graphics community, where it has been used for an impressive interactive photo-editing application [90]. A more heuristic-based version of this same concept is presented recently by Rock et al. [134], who retrieve for their query depthmap a coarsely matching training-set 3D mesh. This mesh is then deformed to match the query scene. This idea bears some similarity to the 'shape sharing' concept of Kim and Grauman [93], as they do not aim for semantic similarity but instead focus on similarity of shape. However, Rock et al. only demonstrate results on a limited set of synthetic, isolated objects. On real scenes, among other issues, their method would rely on a good initial segmentation of the scene.

Both instance-level and generic fitting of 3D models can give a good recovery of missing geometry of that object, but rely on the availability of some form of specific prior model, and this completion method relies on an accurate detector to find and classify each object in the scene. In our approach, by not restricting ourselves to trying to find semantic correspondences to a training set we are able to model the shape of a great range of test scenes without making any assumptions about their semantic makeup. We set out to get as much shape information as possible *without semantics*, thus remaining free of its associated machinery and limitations. In Chapter 6 we do complete scenes by completing individual objects. However, unlike the works discussed above, we do not require a specific training instance for each object. Instead, we *discover* correspondences between objects in different scenes, and by combining these multiple views of objects from different directions we are able to make 3D completions of our scenes.

3.4 Completion using the observed scene

Without a dataset of training models, it is possible to make some completions using data from the observed scene itself. If one is able to accurately detect symmetry, for example, it can be leveraged to complete some types of objects. Law and Aliaga [102] use symmetry to complete partial views. However, they rely on all objects being symmetrical, and they

need user input in the form of the type of symmetry present for each item in the scene. Similarly, [162] and [99] use symmetry to complete models from a single depth view. They both demonstrate results on isolated objects, but not on more complex or cluttered scenes. The constraint of requiring axes of symmetry to both be present and accurately detected is a large limiting factor with these approaches. Using symmetry for object completion is possible, but it is brittle. If symmetry cannot be detected at all, then no prediction can be made. This is the case for example when viewing a cuboid end-on.

The data used to complete missing regions can come from different locations in the same scene, rather than just using planar or rotational symmetry. This approach has been used in many different application areas. In computer graphics, Zheng et al. [179] detect repetitions in building facades to enable missing data to be inferred for each of the detected components. In the robotics community, Chang et al. [20] hypothesise completions of a robot's map of an environment using parts of the already observed world to 'patch in' missing areas of the current map. Harary et al. [68] use a similar approach to mesh completion, finding matches in the existing mesh which could be used to fill in the missing region.

3.5 Using 3D primitives for recognition and completion

Gaining a full dataset of training instances for all the objects present in a scene, and being able to accurately detect them and then fit the training instances to them is very difficult. However, by breaking the scene down into smaller, more generic primitives, it is possible to recognise and model shape without the need for specific training examples. 'Geons' are proposed by Biederman [7] as a set of 3D primitives such as cylinders and cuboids used by humans in their recognition of object shapes (Figure 3.1). Progress has been made on recognising geons in natural scenes, e.g. [174]. In practice, however, this process was found to be challenging due to their "idealized nature" [37], requirement for part segmentation, labelling errors, and the coarseness of features used to extract geons in the first place — see [37] for an overview of these difficulties.

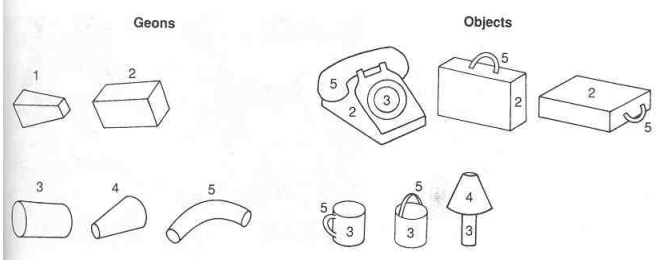


Figure 3.1: Geons (on the left) used used as primitives for describing objects (on the right). Image is from [8].

Bounding boxes While geons have not found success in the computer vision community, the fitting of bounding boxes has become a popular method to explain the arrangement of objects in a scene. Recent work has successfully incorporated high-level information such as gravity and stability [146, 82], and made use of training data to accurately detect bounding box locations [71]. Gupta et al. [64] estimate voxel occupancy from a RGB image, which is regularized using cuboid bounding box hypotheses. The obvious problem with bounding box style methods is that they can only give coarse shape information, which is not suitable for many applications of geometry completion.

Generic deformable models 'Ballooning' meshes has been used with success for inferring 3D shape from 2D images [168, 122]. This system typically works by 'inflating' a mesh under the constraint that it maintains the same silhouette as the 2D input shape, under an assumption of planar symmetry about the image plane. The total volume of the inflated mesh is typically a parameter. This required parameter, plus the symmetry and pose assumptions, are what restricts direct application of this form of system for general purpose reconstructions.

In Chapter 5, we make use of 3D primitives for shape completion. However, unlike geons which are fixed in shape, we learn a distribution of shapes from training data. We are also able to make more fine-grained predictions than bounding boxes; we demonstrate this qualitatively and quantitatively.

3.6 Surface completion

All the approaches of primitive fitting and object detection treat the scene as being constructed of a number of distinct elements. An alternative approach is to directly utilise

the Gestalt theories of object continuation and closure [60] to *continue* the visible surfaces to form *complete* volumes. This concept has been reviewed in some detail by Breckon and Fisher [15], who note that volumes are bounded by surfaces, which are then bounded in image space by contours.

Davis et al. [35] was an early work that completed surfaces by diffusing the signed distance field representation. This is, however, limited to small holes and gaps. To complete larger missing regions Podolak et al. [126] fill holes in a mesh by enforcing watertightness across an octree structure, while Schnabel et al. [144] complete meshes by extending existing surfaces until they meet. This latter approach, in the graphics community, bears some resemblance to recent work in vision by Silberman et al. [154], who tackle the completion of an incomplete multi-view reconstruction as a surface completion problem. By detecting planes they can complete their contours in a 2D projection using a novel CRF method. This method relies both on a piecewise-planar scene, and on beginning with an (incomplete) multi-view reconstruction as input. A good overview of mesh completion algorithms are given in [87].

All of these completion methods are only suitable where the set of missing data is fairly small relative to the size of the observed data. In our case, we are completing the geometry of an unknown surface which is typically at least as large as the observed surface. A further difficulty with completing surfaces from Kinect scans is that successfully *detecting* surfaces in such noisy data is an unsolved problem. Silberman et al. [154] successfully find and complete surfaces, but they limit themselves to planar surfaces, and they assume multiple input views which ensures they have good data to find surfaces from.

More recently, [125] used Gaussian processes to fill in missing data in terrain models of environments, effectively treating terrain mapping as a regression problem. They model their environment as a height field on a 2D grid, which limits its suitability for the more cluttered indoor scenes we are tackling.

3.6.1 Discriminative approaches to voxel completion

We have shown above the difficulties that can come with generative models, with fitting 3D models and primitives, and from using surface completion techniques. In the last few years some discriminative approaches have been used to directly predict the occupancy of

voxels from observed data.

Kim et al. [91] use a conditional random field over a voxel representation of a scene to simultaneously predict semantic labelling of voxels from an RGBD image. For training, they use manually labelled top-down views of the scene. Their final labellings are found from graph cuts over the CRF [14], and their algorithm estimates of visibility and voxel occupancy as part of the solving process. They primarily model the probability that a voxel is occupied by a Gaussian centred on the first observed voxel along a camera ray. High-order-terms in the CRF are used to enforce planar structures and for 'objects' to remain contiguous. However, their method is only evaluated on semantic labelling, using 2D reprojections of the voxel grid into the image and onto the floor plane.

Like [91], Zheng et al. [178] go from a single depth image to a voxel representation of a scene. Their voxelisation algorithm works by completing missing voxels, by extruding visible points in the detected 'Manhattan World' directions of the scene, i.e. in the three orthogonal directions which most closely align with the dominant planes. This part bears resemblance to a similar method used for completions by [99]. This voxel completion by extrusion is limited by the Manhattan World assumption and the extent of visible voxels in the scene.

3.7 Summary

In this chapter, we have examined the range of approaches used to complete missing parts from images, meshes and 3D scenes. We have seen that a wide range of approaches have been shown to work, though many are unsuitable for our purpose. For example, some rely on having a set of objects which can be fitted to objects in the scene [41], while others make assumptions about symmetry [99] or are designed for situations where the missing regions are small relative to the data provided [35].

We note that our work has similarities to some of the previous attempts at similar problems. Our work in Chapter 4 operates using a similar underlying method to Zheng et al., for example. In contrast, though, we are able to make use of supervised learning to learn the prediction of voxel occupancy for each voxel in a scene. To achieve this we develop a novel feature representation for each voxel.

In Chapters 5 and 6 we are able to use *structured* predictions of shape learned directly from training data, in contrast to the more localised models of [91], [99] and [178]. This bears some relation to early work on geons [7], but with the ability to harness a machine learning approach to choose suitable structured elements.

Chapter 4

Learning per-voxel occupancy for scene completion

We showed in the previous chapter the difficulties present with voxel occupancy prediction, and we outlined some prior approaches to this and similar problems. For our problem setup we assume we have a set of voxels of unknown state, and for each of them we wish to infer their value of v. In this chapter we tackle this directly, by devising an algorithm which directly predicts a value for each unknown voxel, given the input depth image $\mathbf{D}$.

We make three key contributions:

1. *Dataset review*: We present a comprehensive review of publicly available datasets featuring RGBD data. As well as being of valuable use in itself, we use this review to motivate the need for our own, new datasets.

2. *Datasets*: We introduce a synthetic dataset generation method, together with a set of synthetic scenes created from this algorithm. We additionally introduce a tabletop dataset, with input depth images aligned to a fused occupancy volume grid. Both these datasets set a formal standard for evaluating volumetric completion algorithms.

3. *Voxel occupancy feature vectors*: We present a set of novel feature vectors which can be used to predict the occupancy of each voxel in a scene. We demonstrate that, in conjunction with a machine learning pipeline, we can use this feature vector to make good predictions of the occupancy of unseen voxels in both of our datasets.

4.1 Problem statement

Given a depth image $\mathbf{D}$ captured from a camera at world rotation $\mathbf{R}$ and position $\mathbf{t}$, we aim to directly predict the truncated signed distance function (TSDF) occupancy value of each voxel in the voxel grid $\mathbf{V}$. That is, for a voxel at location $\mathbf{q} = (i, j, k)$, we seek a function

$$f : \mathbf{D}, \mathbf{R}, \mathbf{t}, \mathbf{q} \rightarrow \mathbf{V}(\mathbf{q}), \tag{4.1}$$

where $\mathbf{V}(\mathbf{q})$ is the TSDF prediction for voxel at location $\mathbf{q} = (i, j, k)$ as described in Section 2.1. We are also given a set of training data $\{(\mathbf{D}, \mathbf{R}, \mathbf{t}, \mathbf{V})\}$. This takes the form of a set of depth images at known pose, each with corresponding ground truth occupancy grid $\mathbf{V}$.

Without loss of generality we assume that the world origin is at location $(0, 0, 0)$ in $\mathbf{V}$ and that the 'up' direction of the scene, i.e. the opposite direction to gravity, is aligned with direction $(0, 0, 1)$ in $\mathbf{V}$. The direction of gravity can be found from the dominant plane in the scene, which could be recovered using methods such as [75, 21, 153], or from an accelerometer attached to an RGBD sensor. In our case we manually mark on this up direction on our 3D scenes together with the extents of the grid $\mathbf{V}$ within which we with to make a prediction.

4.2 Background

In Chapter 3 we reviewed work related to the general problem of predicting voxel occupancy from image data. Here, we provide an overview of work specific to this chapter's aim of making predictions for the value of each single voxel individually, given a depth image input.

Zheng et al. [178] provide a heuristic-based method to predict the occupancy each voxel from a single depth image. Their method for occupancy prediction is as follows:

1. They first segment their depth image into convex regions roughly corresponding to objects, using a method based on implicit algebraic models [9].

2. For each segment they find the Manhattan-world coordinate system using the method of [57].

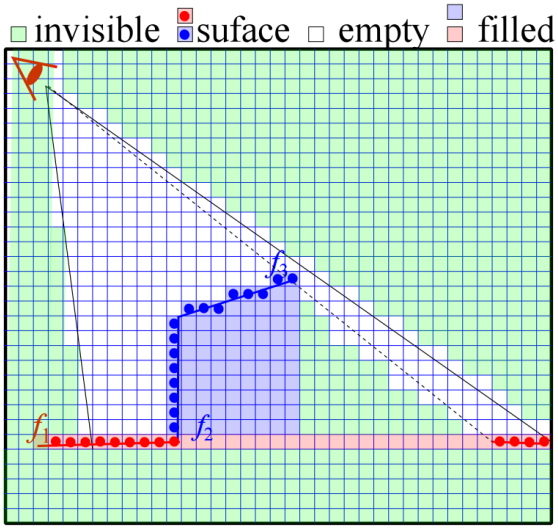


Figure 4.1: An overview of the voxel filling method of Zheng et al. [178]. In this 2D example, each voxel in the light blue region has been marked as 'full' because lines cast in two Manhattan directions have hit a 'surface' voxel. In 3D, a surface voxel must be observed in three Manhattan directions. This image has been adapted from [178].

3. For each voxel within the 3D visual hull of the segment, they cast lines in the 6 Manhattan directions.

4. Where at least $t = 3$ of these lines hit occupied voxels, they mark this voxel as being occupied (Figure 4.1).

This is a simplistic method, but it provides a key insight: A search in $\mathbf{V}$ in the immediate vicinity of position $\mathbf{q}$ is unlikely to provide a clue as to the TSDF value for $\mathbf{V}(\mathbf{q})$. It is instead the locations of *observed surfaces* relative to $\mathbf{q}$ that can be most telling about the true value of $\mathbf{V}(\mathbf{q})$. We use this insight to guide the features we develop for our work. However, not only are our features far more discriminative than those used by Zheng et al., we are able to model a more complex relationship between our feature representation and the label space by exploiting training data.

Kim et al. [91] attempt to predict not only the voxel occupancy of voxels in a scene, but also to predict a dense per-voxel semantic labelling. Their method for occupancy prediction bears some similarity to that of Zheng et al.. The difference is that while Zheng et al. compute distances along *Manhattan axes*, Kim et al. measure distances along *camera rays*. They model the likelihood of a single voxel being occupied as a Gaussian distribution centred on the first occupied voxel along the camera ray. Effectively, their model is built on the hypothesis that voxels that are further behind the depth image,

from the camera viewpoint, are more likely to be classed as empty. While this model of occupancy is naive, by combining this with high-order terms in a CRF they are able to encourage more complex completions. For example, detected planes are encouraged to continue into occluded regions, and detected objects are encouraged to be completed as a single convex shape. Their method suffers from two primary drawbacks: (a) Their occupancy model is conditioned on semantic labels, which is a limiting factor, and (b) their occupancy model is a simplistic Gaussian.

Our aim of per-voxel predictions has similarities with per-pixel predictions, which are popular in images. Typically in labelling problems, pixel data local to a query pixel are used as features to make inference about the target label (e.g. [17]). Our problem differs in one crucial aspect. In image labelling the neighbourhood of a pixel we wish to label is naturally informative about its label, while in our case we wish to make label predictions in 'feature deserts', where the neighbourhood of the query voxel contains very limited information. In this respect, our work bears some relation to pixel labelling which uses *context* and other higher order terms in order to improve results. *Context* has been used with success to improve labelling. This context can come in the form of image-level labels [70] (for example, 'this image is of a street scene'), and on the level of contextual relationships between objects in the scene [164] (for example, 'the region above this region is probably a boat').

4.2.1 Overview of our approach

In our approach, we form a feature representation for each unknown voxel. We can then use training data to learn a model which we can use to infer voxel occupancy from known states.

4.2.2 Features from voxel grids

Compared to the vast amount of feature descriptors for 2D images, depth images and point clouds, features from voxel volumes are rare. In addition to the features of [178] described above, one area in which voxel grids are widely used in conjunction with feature descriptors is for detecting and classifying objects in CT scans for both medical and security applications.

Criminisi et al. [31] use voxel features together with Random Forests for segmentation of voxelised medical image scans, e.g. for segmenting organs out from the body. They use as features a sum over a cuboid of voxels at a specified offset from the voxel of interest.

A good overview of 3D point features used for detecting objects in security scans is given in [52]. These typically accumulate gradients and raw densities of voxels in the neighbourhood of a specified point. Gelfand et al. [58] introduce a point-wise feature for global registration of scans which measures the fraction of a sphere, centred on the surface of a 3D shape, which is occupied by the 3D shape. These existing voxel grid descriptors are ill-suited for our purpose for the reasons described above — in an occluded region of $\mathbf{V}$ the voxel values are unknown, so the immediate neighbourhood of an unknown voxel is typically not informative about its value.

4.2.3 Datasets

To train and evaluate our algorithms, we require a dataset containing scenes with known 3D geometry (in the form of voxel grids), together with depth images captured at known positions relative to the 3D model.

In Table 4.1 we provide a comprehensive list of existing publicly available dataset captured from a 3D scanner of static scenes. We omit datasets captured of humans, for example for action recognition or face and hand tracking. However, a full list including these omitted datasets, together with download links and further descriptions can be found online at [48] and published in [49]. We categorise each dataset according to its *type*:

Turntable datasets Here objects are captured in isolation. By spinning each object on a turntable, a range of azimuth directions are captured, while cameras at different heights capture a range of azimuth angles. Lai et al. [101] introduce a dataset of 300 objects; however, the relative camera locations are not possible to obtain, so 3D models cannot be recovered. Singh et al. [155], on the other hand, capture their data from calibrated registered cameras.

Controlled capture These are scenes where objects are placed by a human arrangements for the purposes of capture (e.g. [2, 61]). This controlled nature of the environment means that properties of algorithms can be carefully studied. Sometimes

		Year	Type[1]	Video	Cam pose[2]	3D model[3]	Labelling
Mian et al.	[113]	2006	C				Object pose
RGBD object dataset	[101]	2011	T	✓			Object
RGBD scenes dataset	[101]	2011	C	✓			Bounding box
Cornell RGBD dataset	[97]	2011	R	✓	✓		Point cloud
NYU v1	[152]	2011	R	✓			Dense[4]
B3DO	[81]	2011	R				Bounding box
Pomerleau et al.	[127]	2011	C	✓	✓✓		
Object segmentation	[133]	2012	C				Object
Willow Garage dataset	[1]	2012	C				Object segmentation
Aldoma et al.	[2]	2012	C				Object pose
Hinterstoisser et al.	[74]	2012	C	✓	✓		Object pose
Meister et al.	[112]	2012	C	✓	✓	✓✓	
DAFT	[61]	2012	C	✓			
TUM Benchmark	[159]	2012	C/R	✓	✓✓		
NYU v2	[153]	2012	R	✓			Dense[4]
Mason et al.	[110]	2012	C	✓			Object
Princeton Tracking B'mark	[157]	2012	R	✓			Bounding box
Karpathy et al.	[89]	2013	R			✓✓	
MSRC RGBD 7-Scenes	[151]	2013	R	✓	✓	✓	
SUN3D	[176]	2013	R	✓		(✓)[6]	Dense[4]
Stanford 3D Scene dataset	[180]	2013	R	✓	✓		
BigBIRD dataset	[155]	2014	T	✓	✓✓	✓✓	Object pose
SHOT dataset	[143]	2014	C				Object pose
RGBD Scenes v2	[100]	2014	C	✓	✓	✓	Point cloud
Mattausch et al.[5]	[111]	2014	R		✓		
ICL-NUIM	[67]	2014	A	✓	✓✓	✓✓	
SUN RGB-D[7]	[156]	2015	R	✓			Dense, 3D bounding box
Rutgers APC RGB-D	[132]	2016	C	✓			Object pose
CoRBS dataset	[172]	2016	C	✓	✓✓	✓	

[1] (T)urntable; (C)ontrolled capture conditions; (R)eal world dataset; (A)rtificial dataset
[2] ✓= Camera pose from RGBD data; ✓✓= Camera pose from external device or calibration
[3] ✓= Many missing surfaces of objects; ✓✓= Data captured with intent of full geometric reconstruction
[4] Dense labelling on a subset of the full dataset
[5] Captured from a Lidar scanner
[6] A limited number of reconstructions are available from [24]
[7] Combines new Kinect v2 frames with new labels on existing datasets [153, 81, 176]

Table 4.1: A review of publicly available datasets captured with a 3D sensing device in indoor environments of static scenes.

these datasets are created to test algorithms designed to work under specific real-world conditions, for example for change detection.

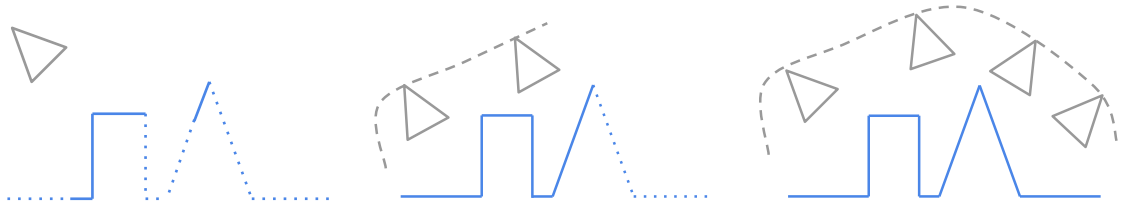
Real world There are scenes which are captured without any modification or movement of objects. For example, Silberman et al. [153] captured single frames and short videos of the interiors of real indoor locations in New York, such as shops, houses and universities. Each image is given dense labelling of object classes (e.g. `bed`, `sofa`), instances (e.g. `bed_1`, `sofa_23`) and type (e.g. `furniture`).

Artificial data The only existing RGBD dataset which is constructed from artificial data is produced by Handa et al. [67]. While their scenes are artificial models, their camera paths are real camera paths taken from human motion of the Kinect. This use of artificial data is an exciting opportunity to get data with true ground truth camera paths, scene geometry and segmentation.

There are two problems that prevent us from using existing 3D artificial datasets, e.g. as provided by [51]. One issue is the problem of where to place the camera; 3D models and scenes do not come with camera positions specified. This therefore requires some algorithmic or manual marking of a 'suitable' camera position for each model or scene. For our synthetic dataset we bypass this issue by fixing the camera viewpoint and ensuring the objects fall into the camera frustrum. However, the more fundamental drawback is that such artificial datasets fail to accurately capture effects of sensor noise, and variations in real-world objects and textures. We therefore additionally capture a real-world dataset which includes these effects.

Camera pose

Knowing the camera pose relative to a 3D world model is vital for our use case. Where video is present, many datasets provide relative camera poses for each frame of the video. Many of these are inferred from the RGBD video data, while others, especially those used for evaluating tracking algorithms, take the camera pose from an external device (e.g. a Vicon Motion Capture system). We note that for the videos which do not have camera pose given we could attempt to find the camera pose using a SLAM system. However, we have found that even with recently released systems e.g. [130] we have encountered many



(a) Many datasets are captured from a single, static viewpoint

(b) In others the camera moves, but does not view all the backs of objects

(c) In our datasets, we move the camera to view all the visible surfaces

Figure 4.2: Different datasets are captured in different styles and from different camera positions.

problems with drift and loss of tracking. We found empirically that such SLAM systems tend to regularly fail on datasets which were not captured with camera tracking in mind.

3D model

A key requirement for training and evaluating our algorithms is a full 3D model. We find that even when camera poses are known, if data is not captured with the aim of capturing and reconstructing the geometric detail, a large quantity of data can be missing from the scans. Consider for example the reconstructed scene from the SUN3D dataset [176] shown in Figure 4.3 (result formed using the reconstruction method of [24]). Many items of furniture are in fact only viewed from one side, as the camera has stayed largely in the centre of the room and been pointed out to the walls. This type of *inside out* capture creates a dataset which is poorly suited to our purpose. We instead require a dataset which has been captured from the *outside in*, in the manner of turntable capture. This ensures that a more full 3D geometry is captured (Figure 4.2)

4.2.4 The motivation for our dataset

The key requirement for our work is a dataset with a good and full *3D model*, together with the relative *camera pose*. This leaves four possible datasets:

1. The BigBIRD turntable dataset [155] contains 600 RGBD images of each of 125 household objects, captured from calibrated, registered cameras. This is a great resource for our work, and indeed we make use of this in evaluating some of our algorithms. However, by only capturing objects in isolation it does not offer the opportunity to evaluate our algorithm on scenes containing clutter and occlusion.

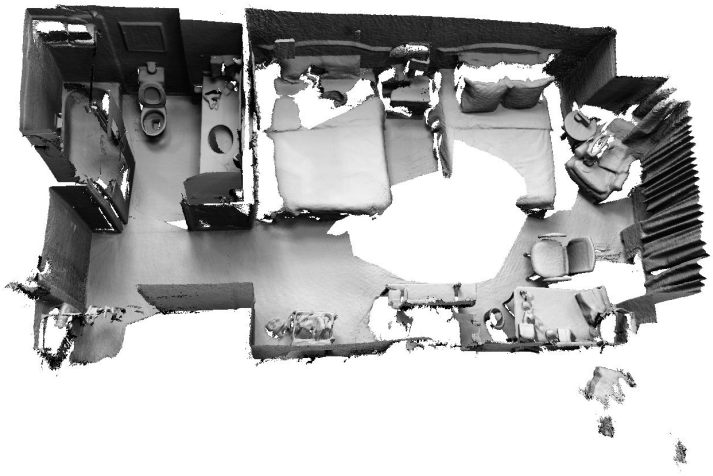


Figure 4.3: A reconstruction from the SUN3D dataset showing the missing geometry due to the *inside out* capture method. Image from `http://redwood-data.org/indoor/models.html`

2. Handa et al.'s [67] artificial dataset is a strong possibility. However, the scenes are captured looking *outwards*, and there are a limited number of frames.

3. Lai et al. [100] is the most suitable for our purpose, with camera poses given for a scenes of household objects where the scene is captured looking inwards, i.e. capturing the arrangement of objects in the round. However, they only provide 11 scenes.

4. Meister et al. [112] do a more comprehensive scan, but only capture three scenes: a statue, a box with some arranged primitives, and an office.

One final option worth mentioning are the two additional labellings of the NYU dataset provided by Kim et al. [91] and Guo et al. [63]. On a top-down reprojection of the 3D points from each frame from the NYU v2 dataset, Kim et al. provide a semantic per-region label. Guo et al., on the other hand, have manually completed each scene with hand-created CAD models, arranged to match the input views. Both of these options suffer from the limitation of being a human-inferred estimation of geometry, rather than representing the true shape.

We conclude from this analysis of existing datasets that we require a new dataset which is focussed on capturing the shape of arrangements of objects. We capture a dataset of

real household objects captured under controlled conditions, plus a synthetic dataset of arrangements of CAD primitives.

4.3 A synthetic dataset for occupancy prediction

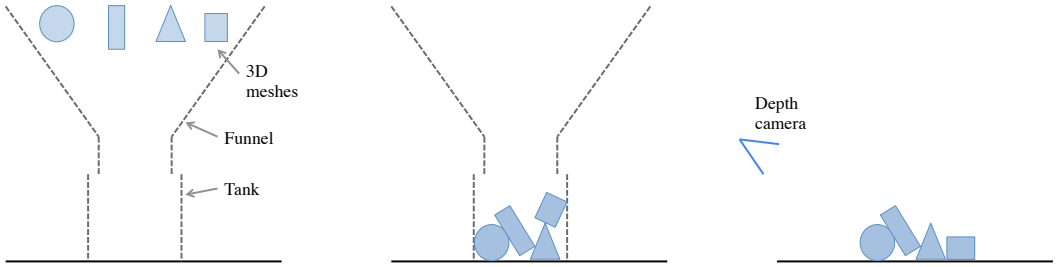
Using synthetic data, as opposed to data captured from the real world, has many advantages. Importantly, the data generated will have excellent ground truth. This enables exact assessment of segmentation, object detection, camera calibration etc. Secondly, we can vary the parameters of the generation and rendering of synthetic data easily. This is very useful to quantify the effect of different capture parameters on the scene.

By generating scenes stochastically, we are able to generate new data as fast as the processor allows. This is in contrast to real-world data, for which collection is typically slow, error-filled and laborious. Finally, it is easy to inspect and critique the creation process of data which is generated stochastically.

Synthetic data has been used with great success for training human body pose estimators [150], performing super-resolution on depth images [107] and evaluating SLAM algorithms [67], to name just a few. For our synthetic dataset we dispense with semantic plausibility in favour of these benefits. We also note that our data is not subject to naturalistic sensor noise.

For our scene generation algorithm, we desire an algorithm to generate random arrangements of objects. We do not require these arrangements to be semantically likely. In addition, by freeing ourselves from a requirement to create scenes which are semantically plausible we can generate far more scenes without restriction. Scenes which are semantically unlikely are in a way the hardest scenes a robot or self-driving car may have to parse, and are correlated with the type of scenes a disaster recovery robot may have to face.

However, we do want scenes to be physically stable and plausible, such that no two objects intersect. We also want the scenes to be diverse, so objects are placed in random arrangements, to maximise the entropy and to make the task challenging. For these scenes we also want all objects to lie within a fixed volume, to ensure that a set of cameras can view all surfaces in the scene. Achieving these aims is non-trivial. In this section we present out approach to achieving this aim.



(a) Initial set up

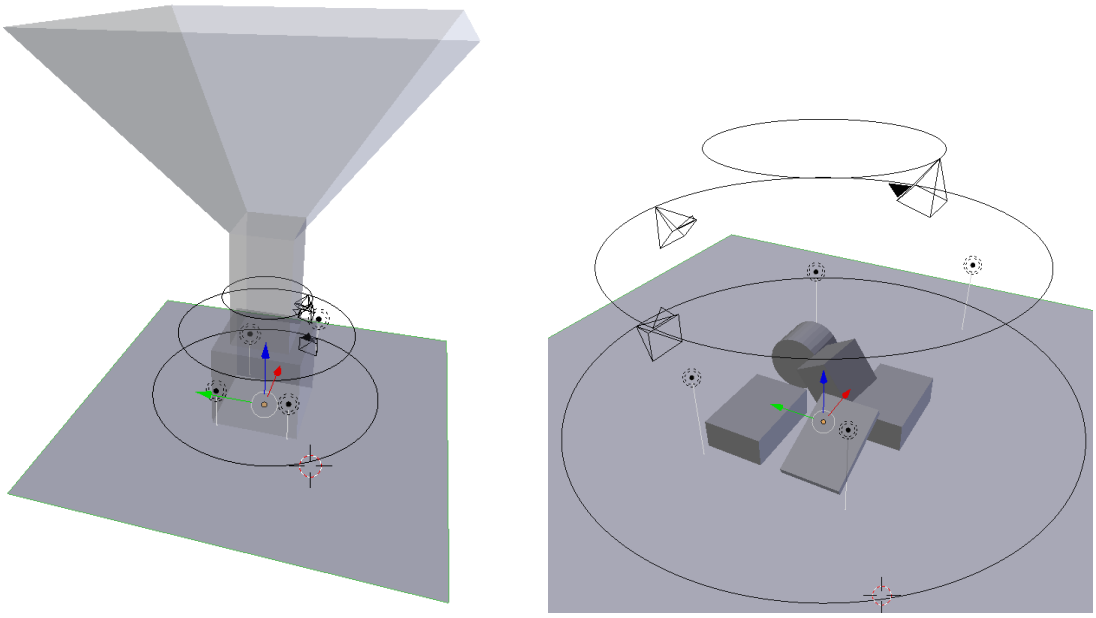
(b) After running physics simulator

(c) After removing the walls and another physics simulator iteration

Figure 4.4: The process for creating and rendering synthetic scenes.

We first randomly select the number of objects to generate. For our work we select a minimum of 5 and a maximum of 15. We select each object randomly from our model dataset with uniform distribution. For our scenes, the model dataset consists simply of cones, cuboids, torii, cylinders and spheres of assorted dimensions. This randomly selected set of shapes are imported into the Blender 3D modelling program and placed at the top of a funnel, as depicted in Figure 4.4(a). We then begin a physics simulator, allowing the objects to drop onto a plane below. We provide walls to prevent objects from 'bouncing' outside the view of the camera. After all the objects have come to a rest, the walls are removed and the physics simulator is run for a second time. This ensures that the final scene is physically stable. Finally, we can take depth renderings of the scene, as depicted in Figure 4.4(b). Figure 4.5 shows the full 3D setup inside Blender.

The full dataset consists of 500 of these synthetically generated scenes. The ground truth TSDF is computed by fusing together the depth images from 42 cameras on a hemisphere viewing each scene. This dataset allows us to easily demonstrate and quantify the effects of various parts of our algorithm in the absence of real world sensor noise. We perform a 60/40 train/test split at a scene level. For each scene we randomly select one of the 42 input views as our training or testing image $\mathbf{D}$. Examples from the dataset are shown in Figure 4.8.



(a) The funnel and ground plane used for creating synthetic scenes in 3D

(b) A set of objects after the physics simulator

Figure 4.5: Screenshots of the synthetic scene generation process in 3D, which was performed using the Blender program.

4.4 A tabletop dataset for occupancy prediction

While our synthetic dataset is large and contains a diverse range of object arrangements, it does have shortcomings. It does not contain real-world sensor noise, and fewer effects such as flying pixels which are present in real-world scans. The objects used are fairly uniform in shape. We therefore motivate a second dataset, captured in the real world.

Our tabletop dataset contains the full geometry of 90 configurations of real objects, reconstructed using the Kinect Fusion implementation of [130]. Examples from the dataset are shown in Figure 4.6, and images from the full set of scenes in dataset are shown in Appendix B. We include statistics of this dataset in Table 4.2, and we show a view of every object from the *training* fold of the dataset in Figure 4.7.

This dataset is seven times larger than the volumetric dataset used in [178]. Each scene comprises between 2 to 6 household objects from a set of 63, placed on a tabletop. We manually annotated the extents of the test volume for each scene, and predictions outside this domain are not used during evaluation. The dataset is split into 60 training and 30 testing scenes, captured in two different locations, where objects present in the testing split do not appear in the training split. We compute estimated camera poses during the

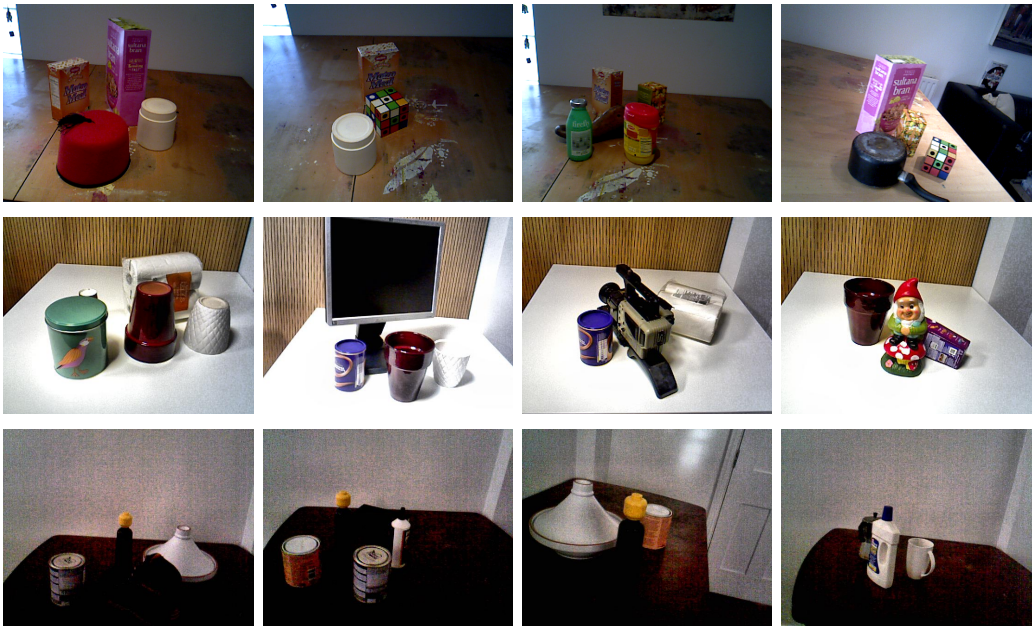


Figure 4.6: Example RGB images from our tabletop dataset. Each row shows a different split from the dataset. Each split was captured in a different physical location, with a different set of objects.

	Training	**Testing**
Number of scenes	60	30
Number of images	240	120
Number of unique objects	37	24
Average objects per scene	4.1	3.9

Table 4.2: Statistics of the tabletop dataset

capture, and use these to form our ground truth TSDF grid. It is worth noting that this ground truth dataset is only accurate up to the reconstruction error of [130].

4.5 Our method for voxel occupancy prediction

To perform occupancy prediction, we first convert our input depth image into a representation which is closer to the voxelised output we desire. We begin with a depth image and an empty voxel grid. In order to simplify the working with these two arrays of different modalities, we convert our input depth image $\mathbf{D}$ into a ternary voxelised representation $\mathbf{T} \in \{-1, 0, 1\}^{I \times J \times K}$. This grid enables features to be extracted in voxel space. Each

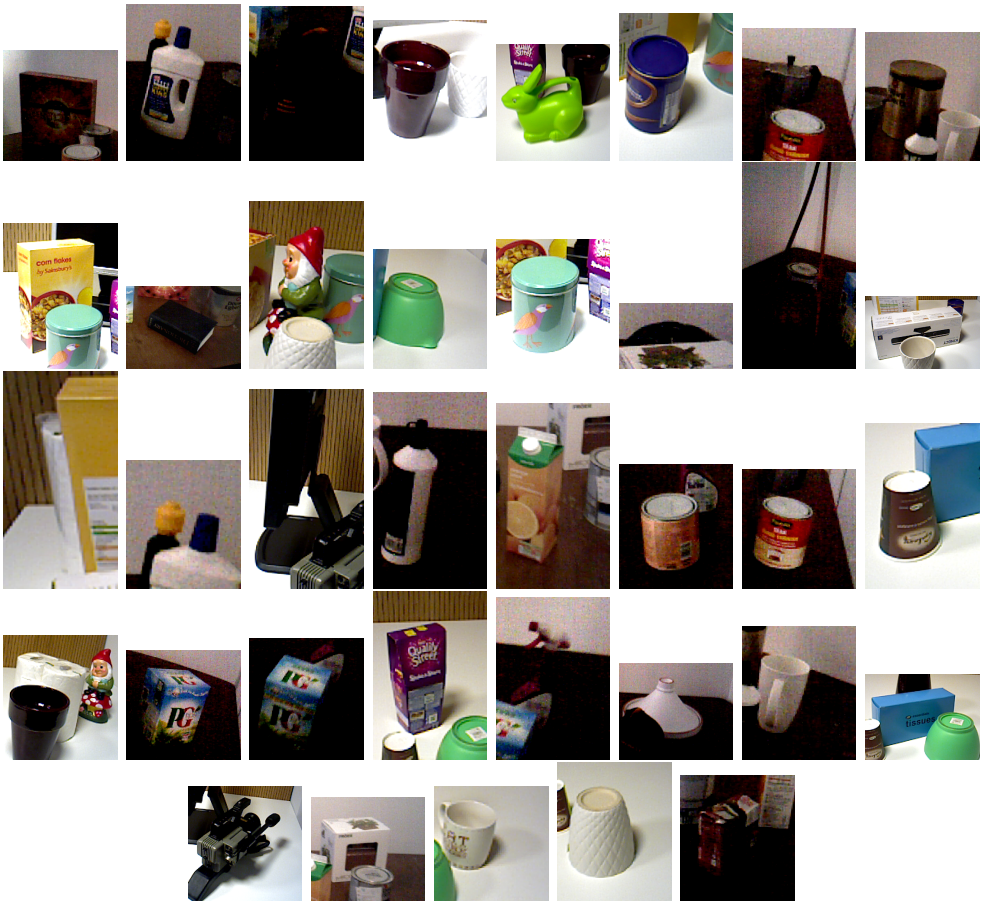


Figure 4.7: All the training objects from the tabletop dataset. See also Appendix B for images of these objects from different views.

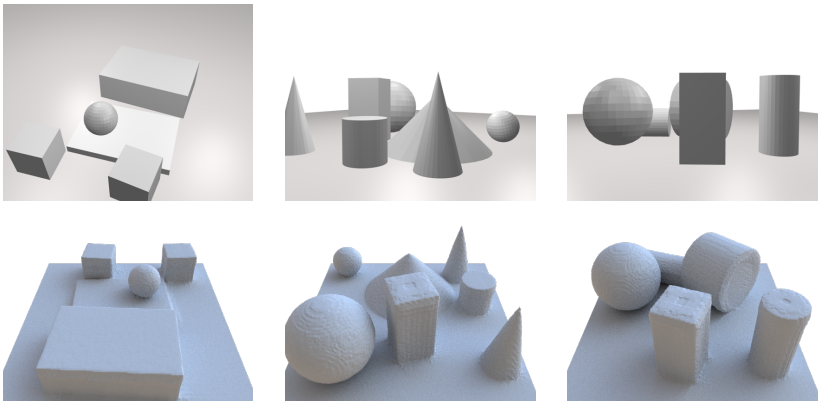


Figure 4.8: Examples from our synthetic dataset. The top row depicts a greyscale render of an input view, used for training or testing a completion model. The bottom row shows a rendering of the ground truth occupancy of each corresponding scene.

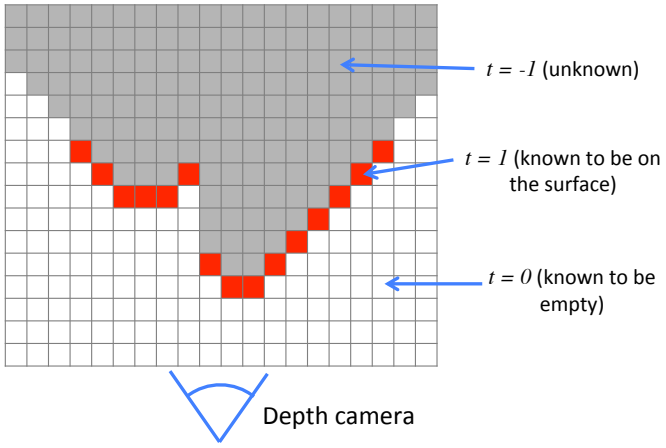


Figure 4.9: Given a depth image and a transformation relative to a grid $\mathbf{V}$, we assign a label to each voxel as described in Section 4.5.

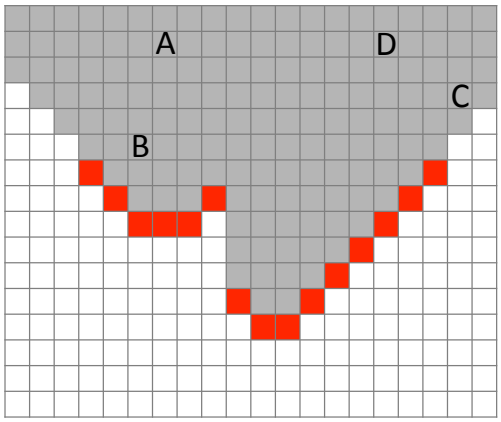


Figure 4.10: An illustration of a hypothesis about voxel occupancy. We hypothesise that voxel A is more likely to be empty than voxel B, because A is further from the region of occupied (red) voxels than B. Furthermore, voxel C is more likely to be empty than voxel D because C is closer to empty, unobserved voxels than D.

voxel $t \in \mathbf{T}$ can take on one of three states:

$$t_{i,j,k} = \begin{cases} 1 & \text{if the voxel is observed to be on a surface} \\ 0 & \text{if the voxel is observed to be empty} \\ -1 & \text{otherwise, i.e. if the voxel is in an unknown state} \end{cases} \tag{4.2}$$

This is shown graphically in Figure 4.9. In our algorithm we aim to predict the true values of the set of unknown voxels $\{\mathbf{q} : \mathbf{T}(\mathbf{q}) = -1, \mathbf{q} \in \mathcal{F}\}$, where $\mathcal{F}$ is the set of voxels which fall inside the camera's frustum.

4.5.1 A hypothesis about voxel occupancy

We hypothesise that a key indicator of the likelihood of a voxel which is in an unknown state to be occupied is given by (a) its position relative to voxels of known states, and (b) what state these voxels of known states are in.

Two examples of this are as follows:

- We suppose that voxels which are closer to occupied voxels are more likely to be occupied themselves. For example, voxel B in Figure 4.10 is more likely to be occupied than voxel A.

- Voxels which are close to empty voxels are more likely to be empty than those which are far from empty voxels. Under this hypothesis, voxel C in Figure 4.10 is more likely to be empty than voxel D.

It is difficult to turn these rather vague hypotheses directly into predictions of occupancy. For example it is difficult to know how close an 'unknown' voxel needs to be to occupied voxels for it to be occupied. Instead, we design features which encode the type of information set out in the hypotheses, and we use our training data to work out a suitable mapping from feature space to an accurate prediction.

We experiment with two types of features to describe each voxel. The first, the Axis-aligned Voxel Occupancy Feature, operates entirely in voxel space. The second, the surface feature, is computed from the depth image, describing the shape of the depth image on the camera ray of the unknown voxel.

4.5.2 Axis-aligned Voxel Occupancy Feature (AVOF)

Our Axis-aligned Voxel Occupancy Feature (AVOF) is used to describe a voxel at position $\mathbf{q}$ in unknown space, i.e. where $\mathbf{T}(\mathbf{q}) = -1$. The feature is computed as follows:

1. From position $\mathbf{q}$ we cast a 'ray' in direction $\mathbf{d}$.

2. We then measure the distance r from $\mathbf{q}$ until the first voxel of a known state (i.e. $t = 0$ or $t = 1$) in direction $\mathbf{d}$. Mathematically, we can write this distance measure

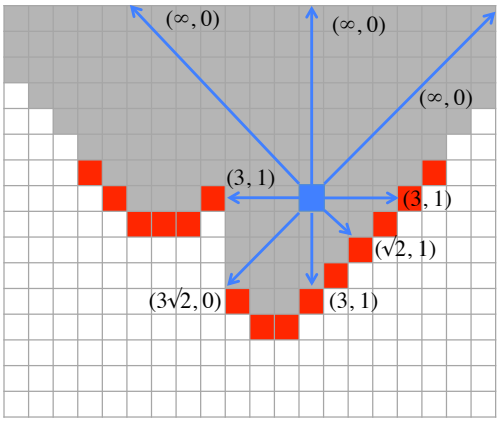


Figure 4.11: The The Axis-aligned Voxel Occupancy Feature shown for a single (blue) target voxel in 2D. Rays are cast from the target voxel in each of the eight compass directions, until they reach a voxel which is of known state. For each ray, the distance to the voxel of known state is recorded, together with the type of voxel it is.

as

$$r = ||\mathbf{d}||_2 \arg\min_{\gamma} \gamma, \tag{4.3}$$

$$\text{subject to} \quad \mathbf{T}(\mathbf{q} + \gamma\mathbf{d}) \geq 0 \quad \text{and} \quad \gamma \in \mathbb{Z}^+. \tag{4.4}$$

We denote the position of this first encountered voxel of known state along the ray as $\mathbf{g}$, i.e. $\mathbf{g} = \mathbf{q} + \gamma\mathbf{d}$.

3. Our feature for direction $\mathbf{d}$ is composed of a tuple combining the distance r and the state of the first encountered voxel: $(r, \mathbf{T}(\mathbf{g}))$.

This process is repeated for a set of directions $\mathcal{D} = \{\mathbf{d}\}$. In our work we set $\mathcal{D}$ to be the cardinal and inter-cardinal directions. In the 2D case this corresponds to the 8 compass directions $(0, 1)$, $(1, 1)$, $(1, 0)$, $(1, -1)$, $(0, -1)$, $(-1, -1)$, $(-1, 0)$, $(-1, 1)$ — see Figure 4.11.

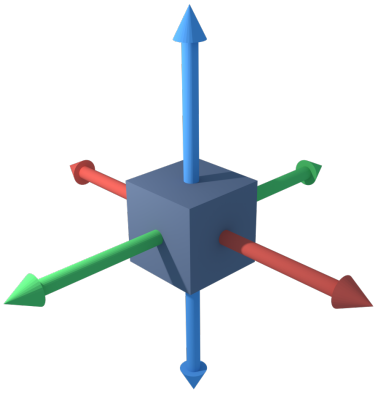
In 3D, the 26 directions can be written mathematically as

$$\mathcal{D} = \{(\Delta_i, \Delta_j, \Delta_k) \quad : \quad \Delta_i, \Delta_j, \Delta_k \in \{-1, 0, 1\}, \quad |\Delta_i| + |\Delta_j| + |\Delta_k| > 0\}. \tag{4.5}$$

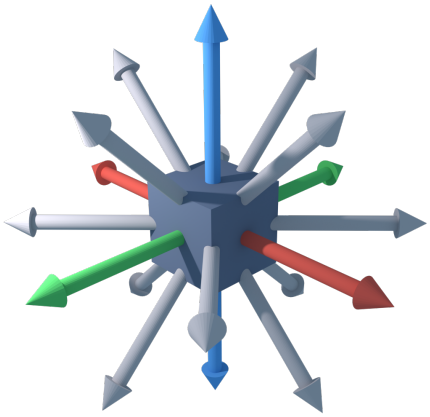
These are illustrated in 3D in Figure 4.12.

This line-casting concept bears some similarity to the features used in RGBD by Drost and Ilic [41]. In their work, they use a point-wise feature which casts a line from a point

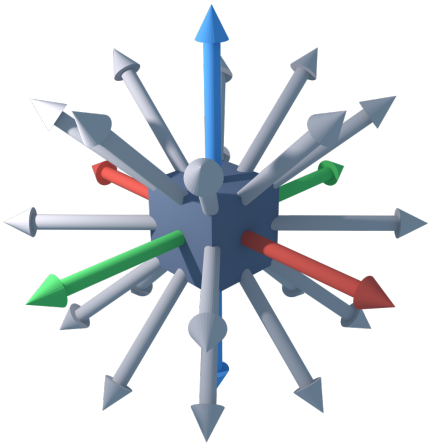
(a) In this figure, we represent our query voxel as a single cube rendered in 3D space



(b) The 6 *cardinal* directions in $\mathcal{D}$ pass from the centre of the cube through each of the 6 faces of the cube



(c) Of the *intercardinal* directions in $\mathcal{D}$, 12 pass from the centre of the cube through the middle of each of the 12 edges



(d) The final 8 intercardinal directions pass from the centre of the cube through each of the 8 vertices

Figure 4.12: The 26 directions $\mathcal{D}$ we search along when forming our feature representation. (a) just shows our query voxel; (b) shows the cardinal directions, while (d) shows the cardinal plus inter-cardinal directions.

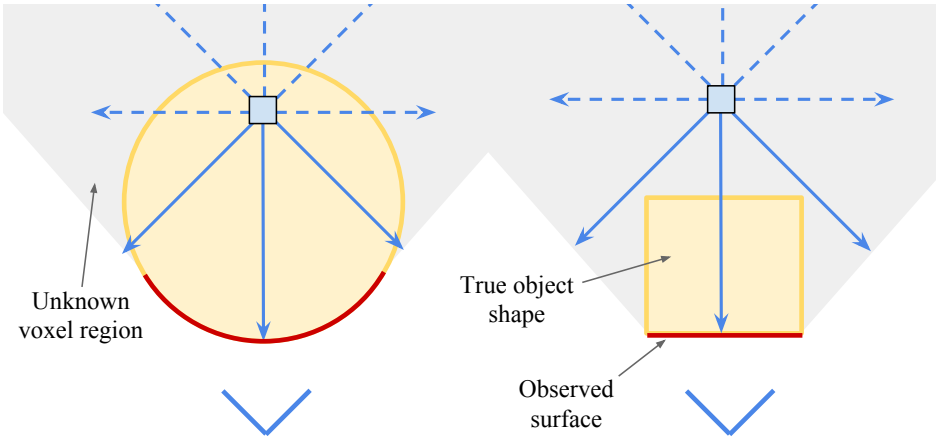


Figure 4.13: Limitations of the AVOF feature representation. The query voxel in the scene on the left has the same feature representation as the voxel in the scene on the right. However, the occupancy of the two is different, which could be deduced from a feature vector which took more information from the visible data from the sensor.

on the image out in a randomly chosen direction. This line stops when it hits a pixel determined to be an 'edge' pixel in the image. Their feature then consists of four metrics encoding the interplay of the cast line, the gradient of the edge image and the 3D normal at the sampled point.

4.5.3 Camera-ray features

We note that the coarse resolution of the directions in $\mathcal{D}$ limits the information content of the AVOF feature. Consider the voxel highlighted in Figure 4.13. The AVOF feature cannot disambiguate between the two different situations, due to the limited number of rays cast relative to the small size of the object.

We could cast rays in more directions to improve the resolution, at the cost of runtime. On the other hand, by analysing the surface *visible in the depth image*, we are able to see that Figure 4.13(a) is a cylinder of a certain radius, while Figure 4.13(b) is more 'boxy'. In fact, the depth image contains many clues about the occupancy of voxels in $\mathbf{V}$. This concept motivates our second feature descriptor $\mathbf{x}_{\texttt{camera\_ray}}$, which is composed of two parts:

$$\mathbf{x}_{\texttt{camera\_ray}} = (\mathbf{x}_{\texttt{surface}}, x_{\texttt{depth}}). \tag{4.6}$$

$\mathbf{x}_{\texttt{surface}}$ describes the shape of the depth image at the pixel $\mathbf{s} = (u, v)$ where the voxel projects to, while $x_{\texttt{depth}}$ is simply a scalar, denoting the distance behind the depth image

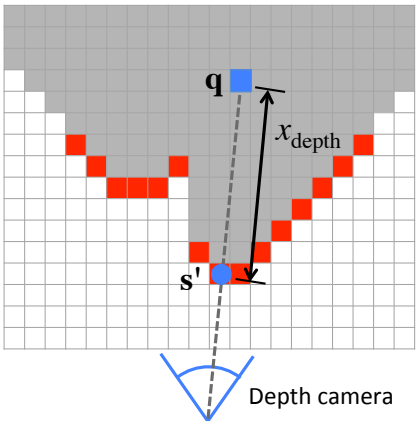


Figure 4.14: How the surface feature is used to make predictions of voxel occupancy for a single query voxel $\mathbf{q}$. Given that $\mathbf{q}$ reprojects to a 3D location $\mathbf{s}'$ on the depth image surface, we represent $\mathbf{q}$ with a descriptor consisting of: a) the shape of the depth image in the neighbourhood of $\mathbf{s}'$, i.e. $\mathbf{x}_{\texttt{surface}}(\mathbf{s})$, and b) the distance x_{depth} between $\mathbf{q}$ and the depth surface along the camera ray.

along the camera ray. A graphical overview of the two parts of this feature vector is shown in Figure 4.14.

Computing $x_{\texttt{depth}}$

We define $\mathbf{s} \in \mathbb{R}^2$ as the point on the image plane which voxel $\mathbf{q}$ reprojects to, and $\mathbf{s}'$ as the 3D reprojection of point $\mathbf{s}$

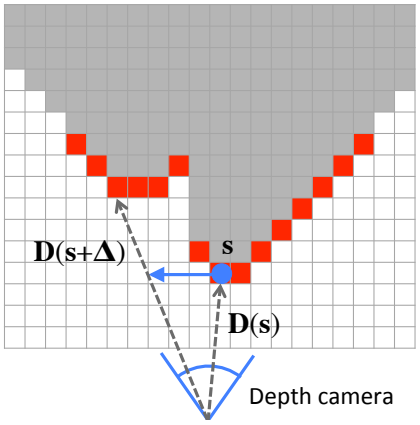
$x_{\texttt{depth}}$ is then simply the distance from the 3D point $\mathbf{s}'$ to voxel $\mathbf{q}$ along the ray of the camera. This is shown in Figure 4.14. We compute $\mathbf{s}'$ using Equation 2.4.

Computing $\mathbf{x}_{\texttt{surface}}$

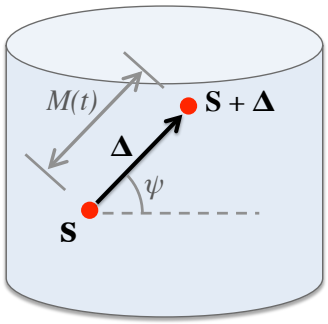
$\mathbf{x}_{\texttt{surface}}$ consists of simple pairwise offset distance comparisons of the input depth image $\mathbf{D}$, used previously with success by [150]. We do not use colour information, instead favouring shape cues provided by the depth image. These *surface* features are fast to compute, and capture the surface shape in the immediate neighbourhood of the point $\mathbf{s}$. Each dimension in $\mathbf{x}_{\texttt{surface}}$ computes the difference between the depth from the camera at pixel $\mathbf{s}$ and the depth at a predetermined 2D offset in pixel space Δ. We define

$$x_{\text{surface}}(\mathbf{s}, \psi, t) = \mathbf{D}(\mathbf{s}) - \mathbf{D}(\mathbf{s} + \Delta), \text{ where} \tag{4.7}$$

$$\Delta = \frac{t \cdot f_c}{\mathbf{D}(\mathbf{s})}(\sin(\psi), \cos(\psi)), \tag{4.8}$$



(a) Surface feature computation on our 2D example scene



(b) How the offsets are computed, from the point of view of the depth image

Figure 4.15: How a single surface feature is computed from a query point **s** on a depth image. The concatentation of multiple such features forms our full descriptor for a point, $\mathbf{x}_{\texttt{surface}}(\mathbf{s})$

where ψ and t are the 'winding angle' and the offset distance respectively. These are parameters of the feature computation, and are defined below.

In 4.8 we are using focal length f_c to map a distance in world space to a pixel offset. This ensures that a fixed offset in pixel space always maps to the same distance in world space, no matter what the depth at that pixel. This helps to ensure invariance in the feature vector.

Figure 4.15 shows this feature vector mapping in a 2D top-down view, and in the space of the depth image.

For a single point $\mathbf{s}$, we compute $x_{\texttt{surface}}(\mathbf{s}, \psi, t)$ for a range of different values of ψ and t, and concatenate the results into the full feature vector $\mathbf{x}_{\texttt{surface}}$. For our experiments we use $\psi,\ t \in \{0°, 45°, \ldots, 315°\} \times \{0.01m, 0.02m, \ldots 0.10m\}$, where m is meters, resulting in an 80-dimensional feature.

4.5.4 Our trained model

Regardless of which of our features described above are used, we train a regression Random Forest to make the mapping from our feature vector at a voxel to the occupancy prediction at that same voxel, i.e. $\mathbf{x} \rightarrow v$.

A Random Forest is a set of decision trees, each of which is trained on a different subset of the training data. During training, at each node we choose the axis-aligned split

which gives the highest reduction in mean squared error in label space of the target values — see e.g. [16] for more details. At test time, for each voxel each tree in the forest returns a prediction from a leaf node. We take the mean of all these predictions together to get our final TSDF prediction for each voxel.

Random forests are well suited for this purpose, as they are fast at test time, easy to parallelise, and most importantly are robust to noisy data and out-of-range features. They automatically select the features most suitable for separating the classes (unlike e.g. SVMs) and they examine each dimension separately, making them robust to different scaling of each feature. This suits our application where features come from different data modalities. They are also inherently non-linear.

Forest details For our experiments we use a forest with 100 trees, and we train each tree to a maximum depth of 30. For tractability we train our forest with a randomly selected subset n of the total set of voxels available for training from all the scenes. For our experiments we use $n = 200,000$, and we examine the effect of varying n in Section 4.6.3.

4.6 Evaluation

In our evaluation we answer three questions:

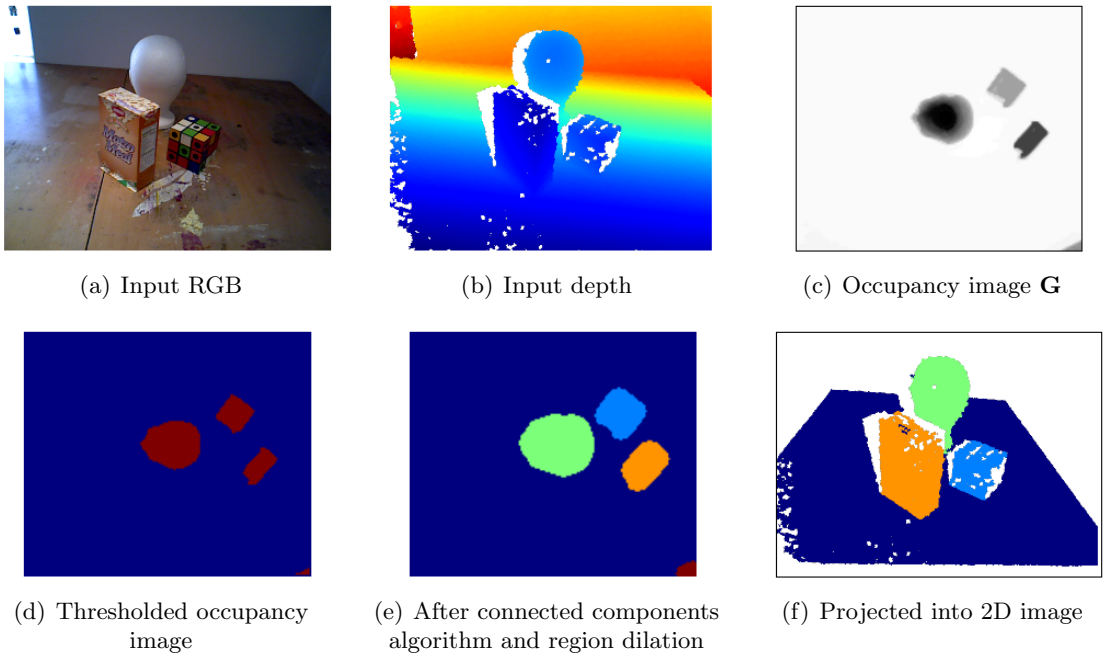
1. Which feature variant gives the best performance, from the choices of Axis-aligned Voxel Occupancy Feature, camera-ray features or a combination of the two?

2. How do these variants perform in comparison to baselines?

3. What are the qualities of the predictions made from our algorithm?

4.6.1 Baseline

As a baseline comparison we re-implemented the method of Zheng et al. [178]. This recently published method is a state-of-the-art heuristic-driven approach to voxel completion, which we have described in Section 4.2. They first segment their image into regions, before forming a voxel completion for each region separately.



(a) Input RGB (b) Input depth (c) Occupancy image **G**

(d) Thresholded occupancy image (e) After connected components algorithm and region dilation (f) Projected into 2D image

Figure 4.16: The pipeline used for segmentation for the baseline implementation.

We found their segmentation method to be very difficult to reproduce as it is a complex method depending on several previous publications, with some unstated or implicit parameters. We therefore give a very conservative comparison by using a segmentation method which has access to the ground truth occupancy grid. This method is limited to situations where objects have some separation on the 2D plane. As this is the case for the majority of our tabletop dataset this method works well.

1. We first create an overhead occupancy image $\mathbf{G} \in \mathbb{Z}_{+}^{I \times J}$. Each element $G_{i,j}$ is given by the count of occupied voxels in each slice of the the ith row and jth column of $\mathbf{V}$, i.e.

$$G_{i,j} = \sum_{k=0}^{K} [v_{i,j,k} < 0]. \tag{4.9}$$

2. Next we threshold this occupancy image to create a binary image, where each pixel in the binary image is 1 where $G_{i,j} > p$ and 0 otherwise. We set $p = 5$.

3. We then perform connected components algorithm on this binary image, thus giving a unique label to each segment in the image. Each unique label is dilated used a disk-shaped element of size 3. This is to mitigate against erroneous misalignments between the voxel grid and the depth image, caused by failures in the Kinect regis-

tration algorithm.

4. These labels are transferred back to the full voxel grid, so each occupied voxel is therefore given a label.

5. Finally this labelling is projected back into image space to give a per-pixel labelling.

Example results from this pipeline are shown in Figure 4.16.

In their description of their voxel completion algorithm, Zheng et al. implicitly fix the parameter $t = 3$ (see Section 4.2). In our reimplementation we allow for different values of t, and for our experiments we use $t = 2$ and $t = 3$.

4.6.2 Evaluation on the synthetic dataset

In Table 4.3 we present quantitative results from the synthetic dataset. We note that the combination of both the AVOF (Axis-aligned Voxel Occupancy Feature) and the surface features gives the best overall performance, however, using using AVOF alone gives an almost comparable performance.

We present qualitative results from this dataset in Figure 4.17. The first column shows a greyscale rendering of the test scene, from the viewpoint which the depth image is captured from. In the second column, we show a 3D rendering of the surfaces captured by the depth camera, as observed from a viewpoint 90 degrees to the right of the original image. The third column shows our predicted completion, viewed from the same angle as the second column. The final column shows the ground truth, formed from the fusion of all the depth images captured of the scene. This is a best-case reconstruction we could expect if we viewed the scene from multiple angles.

It is clear that on the simpler scenes, e.g. (a) and (b), we form a successful completion with a high similarity to the ground truth data. Shapes with less than half of their surfaces visible, such as the sphere in Figure 4.17(a), are completed to a very good standard. We can also observe some more complex interactions in the predictions for more challenging scenes. Figure 4.17(f) contains many objects at different angles, and the algorithm has managed to reasonably successfully recover a region of heavily free space underneath the larger tilted cuboid. While the prediction is noisy and lacks the true structure of the

	IoU	Precision	Recall
Surface	0.716	0.870	0.797
AVOF	0.769	0.893	**0.838**
AVOF+ Surface	**0.771**	0.906	0.831
[178] (t=2)	0.645	0.931	0.677
[178] (t=3)	0.322	**0.968**	0.326

Table 4.3: Quantitative evaluation of the implicit prediction algorithm on our **synthetic dataset**.

ground truth, for an application such as robot grasping, this would likely give enough information to plan a suitable robot arm path.

On more complex scenes the results look less similar to the ground truth. This can occur for a number of reasons. Where there is a large amount of occlusion, the system can make big failures due to the uncertainty in prediction. See for example Figure 4.17(g), where the inverted cone blocks a large region of space from the camera's line of sight. The proposed completion is very uncertain and noisy, as the per-voxel feature vector can give very little definite information for each voxel. Furthermore, where only a small part of shapes are observed, the completion can also be poor. This is apparent in Figure 4.17(e), where only a section of the larger sphere is visible. The limit to the information captured in the feature vectors means that in these cases, a smooth and accurate completion cannot be expected.

In Figure 4.18 we show a final set of results comparing our approach to our baselines on a challenging scene. There are three points to note about this scene, which contribute to making it deceptively challenging:

1. The camera is at a low angle of elevation, so the tops of objects are largely unseen.

2. There is significant occlusion, in particular of the large cylinder on the right hand side of Figure 4.18(c).

3. The cuboid at the front of the scene has only one face visible by the camera.

Here, we can see how our method (Figure 4.18(f)) forms a noisy prediction, but one which is globally representative of the ground truth. In particular, the algorithm has made reasonably sensible estimates of the shape of the foreground cuboid and the upright cylinder. The background cylinder has not had its shape recovered, but the mass of where

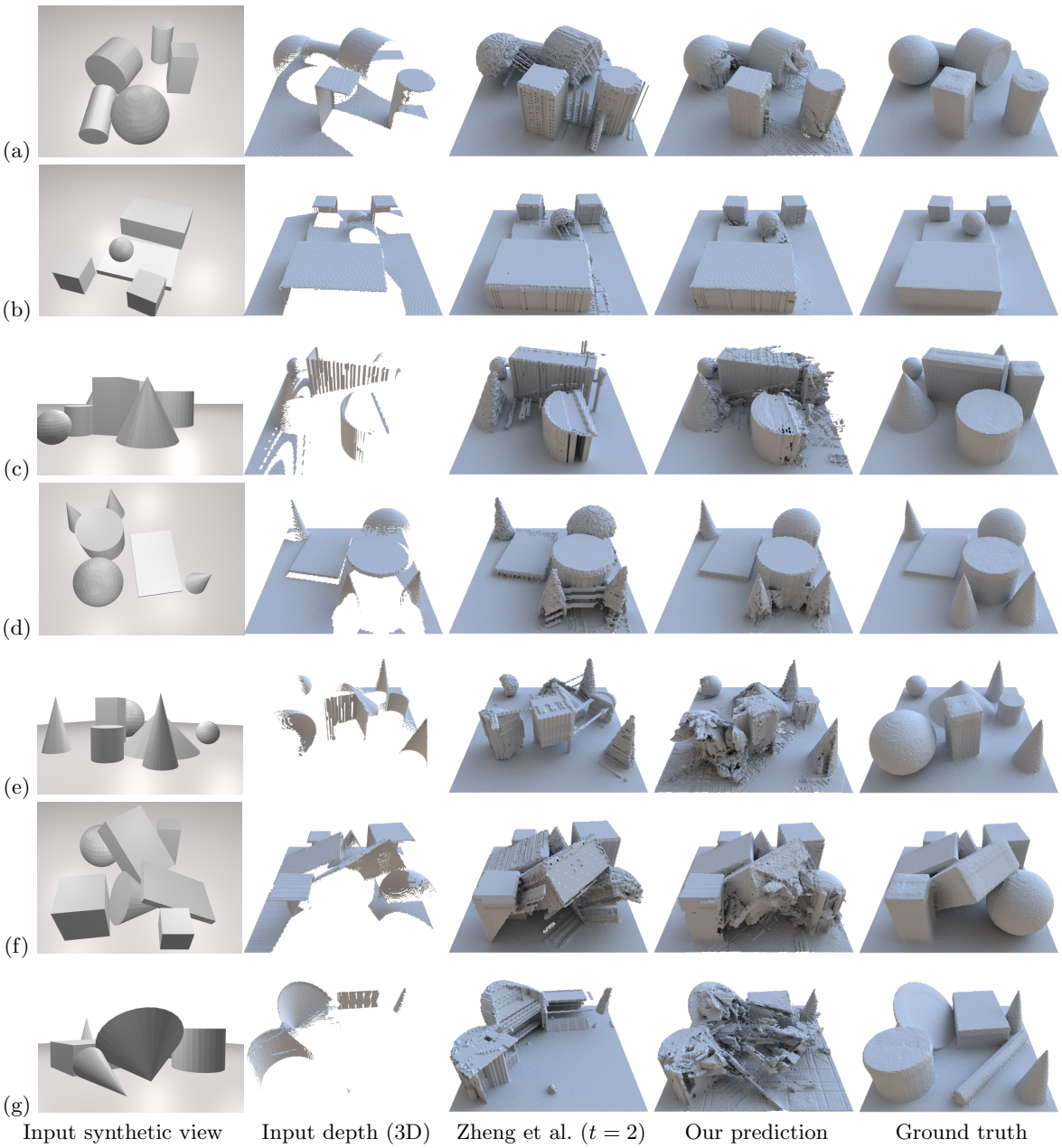


Figure 4.17: Qualitative results from the implicit predictions on the synthetic dataset.

occupied voxels are located is broadly correct. Both implementations of the baseline under-predict volume, only managing to make predictions within the convex hull of each segment. All algorithms, however, miss the sideways completion of the smaller horizontal cylinder.

We note that for our data, using $t = 2$ gives a higher performance than $t = 3$ for the baseline algorithm. This lower value for t vastly improves recall (Table 4.3) with only a small hit to precision. Figure 4.18 shows this effect well. As we note, both the baselines fail to recover most of the mass of the objects. However, the baseline with $t = 2$ has a larger amount of volume recovered than the $t = 3$ variant, as it has a less restrictive requirement for completion.

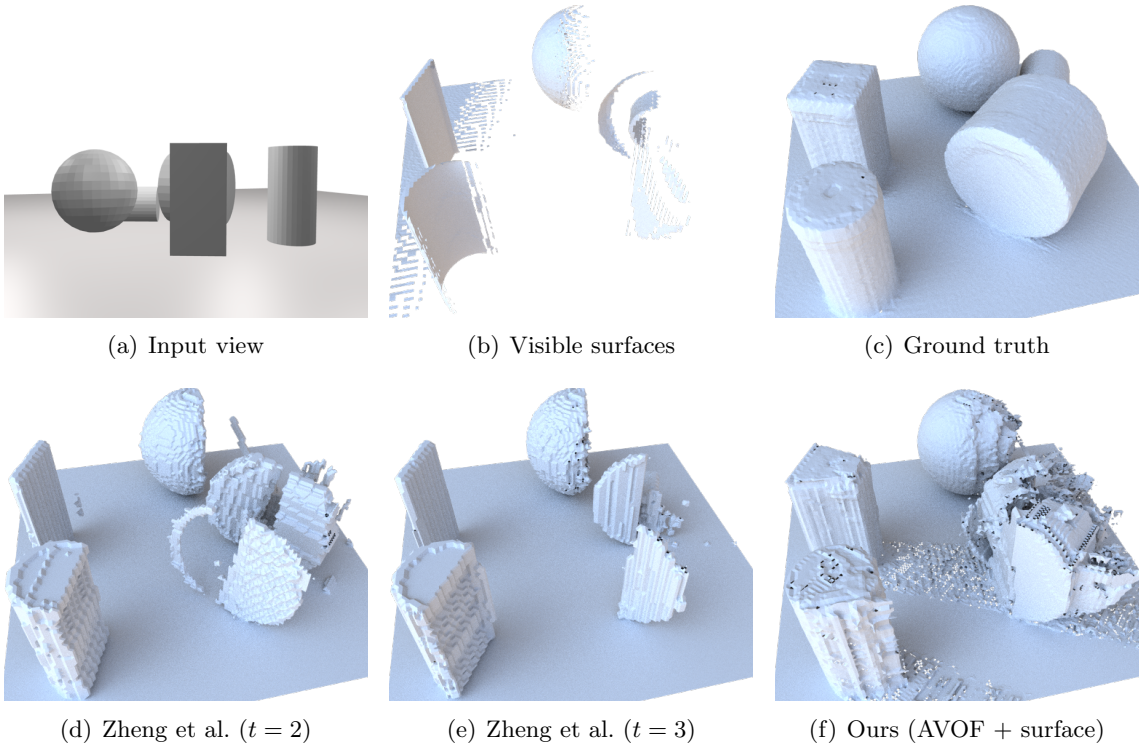
Our algorithm is able to complete more of items such as the cylinder than the baseline for two reasons. Firstly, it has access to a richer description of the world, with a higher dimensional feature vector computed from the depth image and the points in 3D space. Secondly, our use of supervised machine learning means we are able to capture more subtle relationships between our features and the output space than can be gained from a simple thresholding.

4.6.3 Evaluation on the tabletop dataset

In Table 4.4 we present quantitative results on our tabletop dataset. We compare three different combinations of feature vectors, and two variations of the baseline algorithm from [178].

On the intersection over union (IoU) score, our full algorithm performs better than all other variants, and also both versions of the baseline algorithm. Using the AVOF alone also performs better than the baselines. While the surface feature alone performs worse than the baseline with $t = 2$, we note that the baselines were implemented with access to ground truth segmentation.

Our full system gains a very high rate of precision (0.823), although the $t = 3$ baseline gains a higher rate of precision (0.853). The low rate of recall from this baseline (0.405) suggests that the high precision is achieved at the expense of a large under-prediction of volume



(a) Input view (b) Visible surfaces (c) Ground truth

(d) Zheng et al. ($t = 2$) (e) Zheng et al. ($t = 3$) (f) Ours (AVOF + surface)

Figure 4.18: A comparison of our algorithm against the baselines on one scene from the synthetic dataset

Effect of training size A key variable in the algorithm is the effect of the amount of training data. In particular, we would like to answer the question: *Are the failures in the predictions caused by a lack of training data, or are they caused by a fundamental limitation of the algorithm?* Figure 4.19 shows how the IoU score varies as the size of the training set is varied, from using just 10 voxels from the training set, up to using 10^6 training voxels. For each run of the experiment, we randomly sample each voxel, without replacement, from the entire training set. This experiment was run on the tabletop dataset, using a combination of AVOF and surface features. It is apparent, on the linear scale, that the graph levels off at approximately 5×10^5 training examples. This suggests that the scores we are achieving are an upper limit on what is possible given this algorithm. It is also apparent that we can get a good performance with relatively few training examples. We discuss potential modifications to the algorithm, which may improve the scores further, in Section 4.7.1.

	IoU	Precision	Recall
Surface	0.522	0.722	0.666
AVOF	0.625	0.819	**0.732**
AVOF + Surface	**0.627**	0.823	0.729
[178] (t=2)	0.528	0.773	0.630
[178] (t=3)	0.378	**0.853**	0.405

Table 4.4: Quantitative evaluation of the implicit prediction algorithm, evaluated on our **tabletop dataset**.

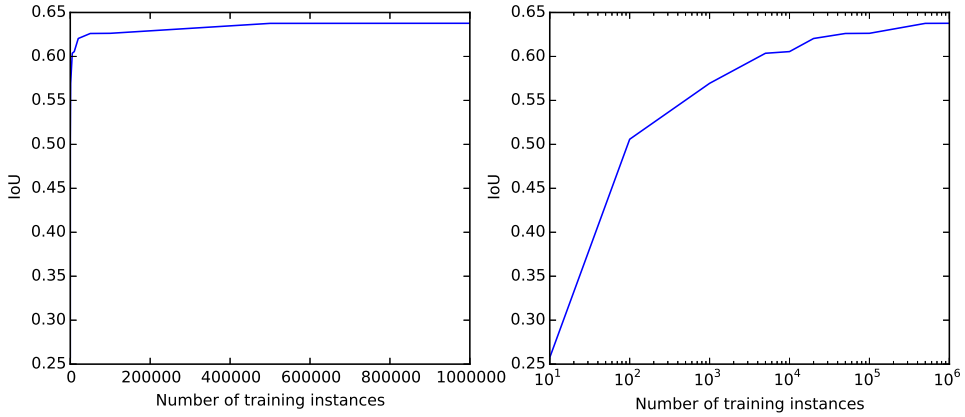


Figure 4.19: Performance of the per-voxel prediction model as the number of training voxels increases on the tabletop dataset. We show the number of training voxels on both a linear scale (left) and a logarithmic scale (right).

Implementation and timing details Our framework is largely written in Python, with the feature computation and Random Forest in C with Python wrappers. Computing the AVOF features for a single voxel grid of size $128 \times 128 \times 128$ takes around 18s, and the camera-ray features take around 2s for a single image. Making a prediction using the forest takes around 1.5s for a single scene. Making the algorithm real-time is an attractive future possibility. Two ways this could be approached are:

- ⇛ Fewer features could be computed, enabling faster processing potentially at the expense of accuracy
- ⇛ The existing features could be computed with a more efficient implementation, potentially exploiting GPGPU programming.

We leave these options for future work.

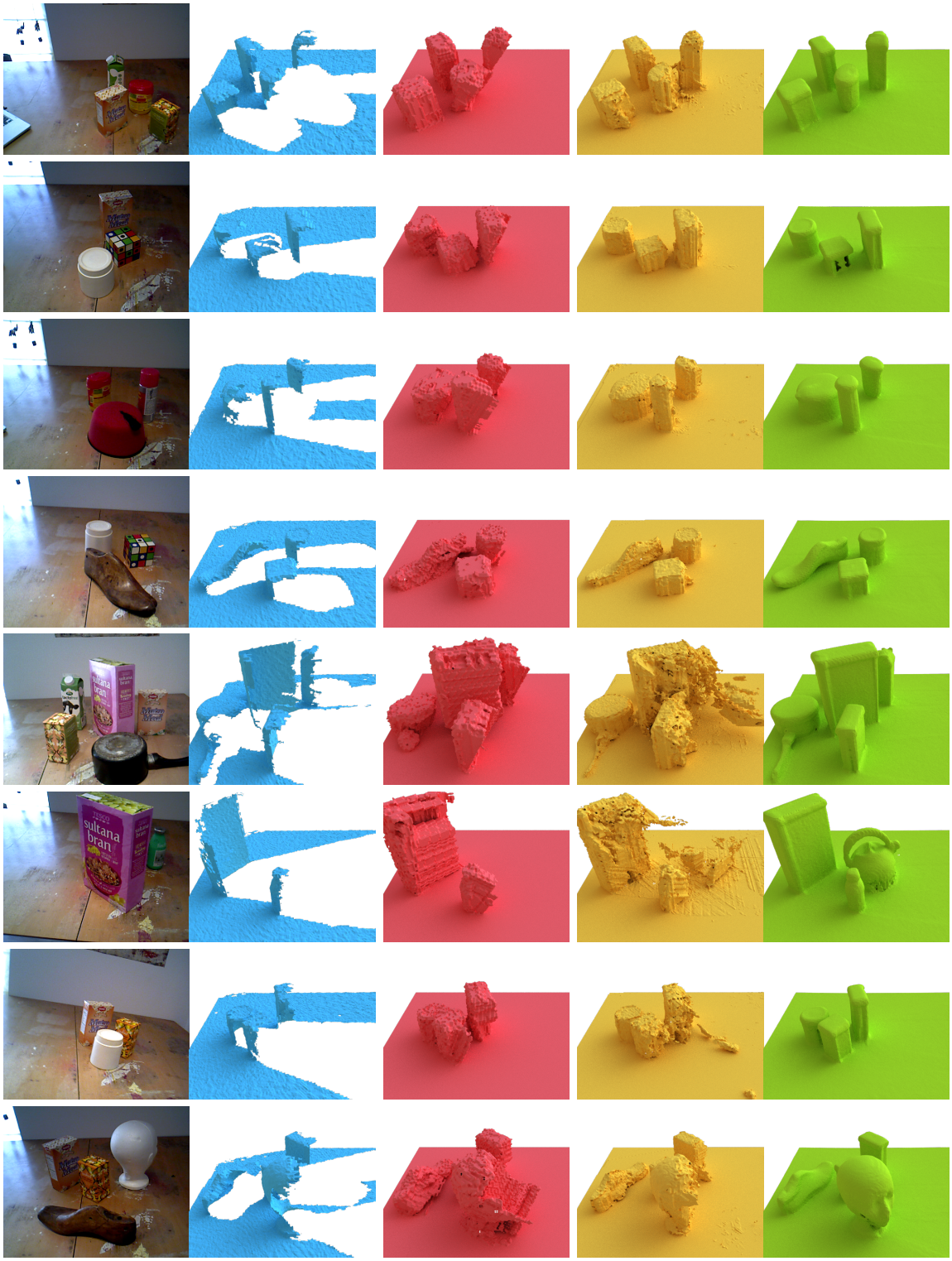


Figure 4.20: Qualitative results from the implicit predictions on the tabletop dataset.

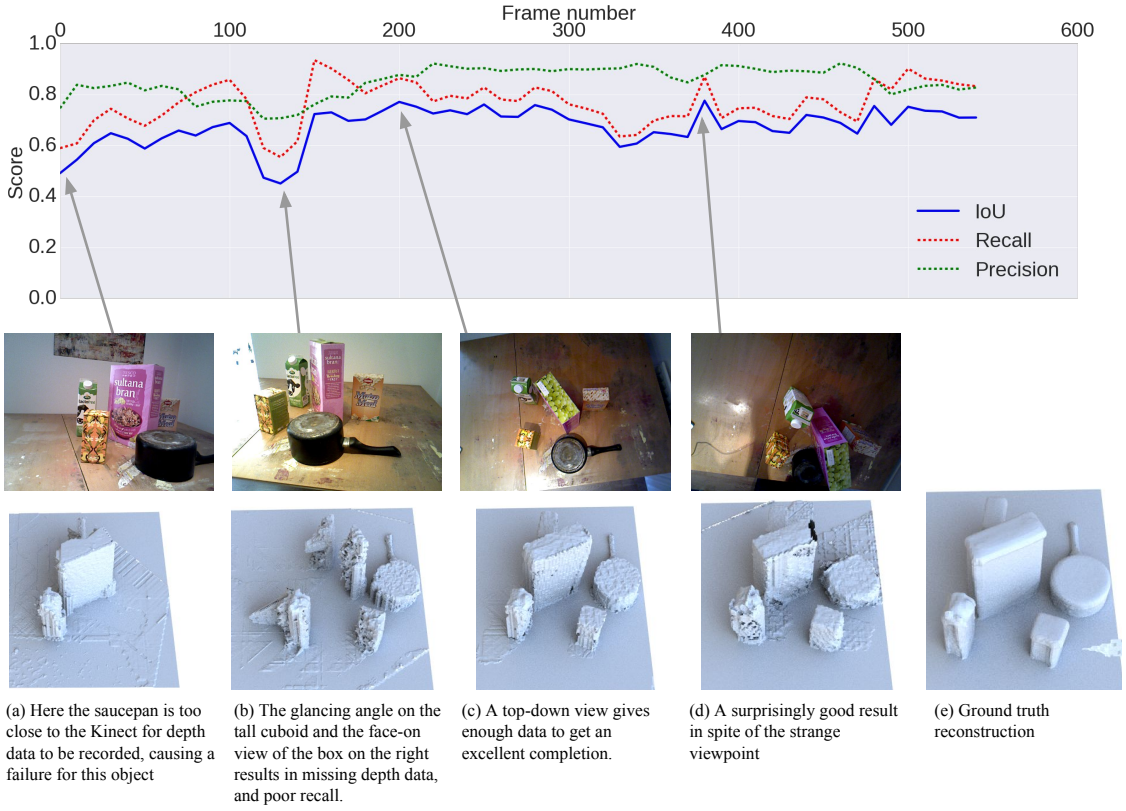


Figure 4.21: Sensitivity of the completion to changes in camera viewpoint, for a single video sequence from the tabletop dataset. (a)-(d) show completions from different camera viewpoints. Each completion is made independently, using only a single frame as input. The ground truth completion is shown in (e).

4.6.4 Sensitivity to camera viewpoint

In this experiment we examine how sensitive the algorithm is, at test time, to the specific camera viewpoint used as input. For this we select a single test video from the tabletop dataset, which consists of 563 frames. We independently make a prediction for every 10th frame in the sequence, and evaluate precision, recall and IoU compared to the ground truth. These results are shown in Figure 4.21. Below the graph we show the input camera views and completions for four points of interest over the sequence. We can see that, in general, our algorithm produces fairly stable predictions over a variety of camera poses. This includes later images in the sequence where the camera is pointing backwards at the scene from above. We still manage to perform a good completion even from this unusual viewpoint. However, where occlusion is large, or depth data is missing, we tend to under-complete objects, hurting recall, IoU and the plausibility of completion.

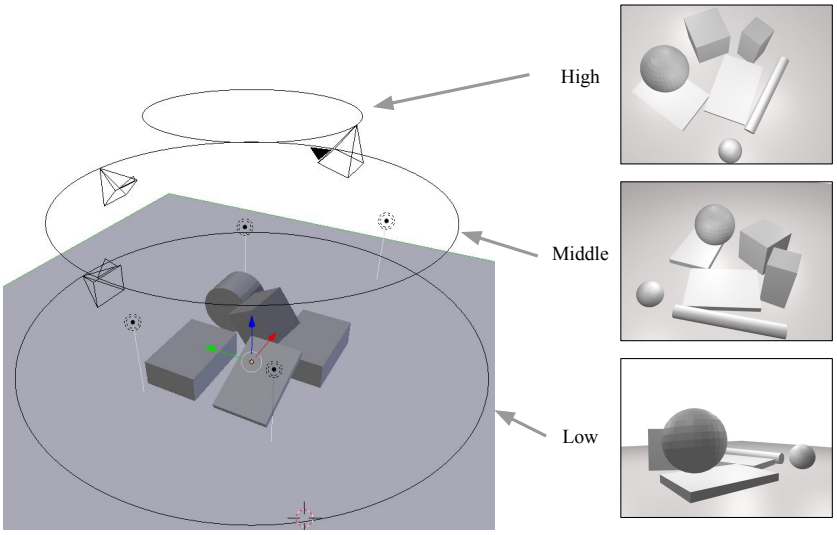


Figure 4.22: The different angles of elevation in the synthetic dataset

	IoU	Precision	Recall
High	0.928	0.956	0.970
Middle	0.874	0.940	0.925
Low	0.670	0.872	0.740

Table 4.5: An analysis of how the accuracy varies for images captured from different angles of elevation in the synthetic dataset. See Figure 4.22 for an illustration of the different viewpoints used.

4.6.5 Synthetic dataset viewpoint performance

Each images in the synthetic dataset is captured from one of three angles of elevation. We classify these here as 'High', 'Middle' and 'Low', as shown in Figure 4.22. To gain an understanding of how the algorithm performs under different viewpoints, we can find the average score for all images captured from each viewpoint. These results are shown in Table 4.5. We can see that all scores decrease as the camera moves closer to the ground. This is to be expected, as there is far more occlusion in these images, and therefore more uncertainty.

4.6.6 Per object performance

An important part of the introspection of the performance of this algorithm is to look at the quality of completions of *each object*. We have already seen qualitatively how different objects perform in Figures 4.20, 4.18 and 4.17. Quantitatively evaluating the per-object performance is difficult as we only have data with each object in scenes surrounded by

	Object	IoU	Precision	Recall
	Whitecup	0.686	0.823	0.813
	Shoe	0.673	0.860	0.759
	Redbottle	0.670	0.858	0.765
	Fez	0.648	0.880	0.711
		...		
	Head	0.590	0.833	0.669
	Purple cereal	0.578	0.798	0.672
	Green bottle	0.546	0.801	0.638
	Red bottle	0.545	0.777	0.646

Table 4.6: The four best and four worst performing objects with the per-voxel algorithm, on the tabletop dataset. For each object, we give the average score of all the scenes which it appears in.

other items. As an example of this difficulty, consider the case of a single voxel which has been falsely labelled as being occupied. It is hard to know which object in the scene is to 'blame' for this false positive — in fact, it may be a combination of multiple objects.

To deal with this difficulty, we present per-object results on the tabletop dataset by averaging together the scores for each *scene* that object appears in. Because each object appears randomly in several different scenes, by averaging we are able to eliminate some of the effects of the other objects in the scene. These per-object results are shown in Table 4.6. We can see that some of the more challenging shapes do surprisingly well, such as the show and the cylindrical white cup. Larger objects, such as the purple cereal box and the head tend to do less well.

4.7 Conclusions

In this chapter we have presented a method to make predictions for the occupancy of unknown voxels from a single depth image. Our algorithm provides excellent results in simple scenes with limited occlusion, outperforming our baseline algorithm. We present one novel feature and an adaptation of an existing feature. While these work best when used in combination, we found that just using our novel AVOF feature gives almost as high results. There are very few parameters to adjust in our algorithm, meaning that little adjustment is required to apply to a new test domain.

In our work we have assumed that we have access to training data, yet in contrast to many supervised approaches we do not need expensive, hand-labelled training data. Instead, we just need depth images aligned to a volume grid of ground truth occupancy values. As reconstruction algorithms become more reliable, and depth sensing becomes more prevalent in the real world, we expect that the availability of such training data will increase. Furthermore, we have presented two new datasets which can be used to train and evaluate voxel occupancy prediction methods.

We note that the system could be trained and tested on voxel grids from any point in a fusion pipeline, e.g. at any point in time on a grid as reconstructed by [80]. This presents an interesting opportunity for scan completion, where our prediction becomes replaced as more data arrives.

4.7.1 Limitations and opportunities for future work

While the scores for our method are high, we note that qualitatively, the predictions are noisy and appear to be ill-formed. This is a natural result of making predictions independently for each voxel; with no prior over the final result, we can end up with many adjacent voxels have widely different final TSDF values even if this is unrepresentative of the training data. We consider three options for future work which may improve this aspect of the results:

1. We could regularise the predictions from the forest, using a spatial prior. A typical method for this would be to use a Markov Random Field (MRF). In an MRF the spatial prior may be encoded as pairwise potentials between voxels, encouraging

neighbouring voxels to take on similar values.

2. Alternatively, we could make use of recent work on *entangled forests* [116] to encode more contextual information into the final result. In an entangled forest, an initial forest makes a prediction at each target location. After this, a second forest is used to make the final prediction. The second forest has access to the original feature vector used to train the first forest, in addition to the labels predicted by the first forest at offsets relative to the target location. This means that the final prediction is able to take into account a greater degree of spatial information than a standard forest. These were used with success for voxel segmentation by Montillo et al. [116].

3. A final option would be to make use of *structured* machine learning. In structured machine learning, a multivariate prediction is made for each input feature vector, in contrast to the scalar predictions we use in this chapter. By making predictions which already have plausible spatial structure 'built-in', the final predictions tend to be more regularised than classic univariate prediction.

Both options 1 and 2 are forms of regularisation of the results of our current algorithm. This means that their final predictions are still limited by the feature vector representation of each unobserved voxel. On the other hand, structured machine learning allows for predictions with plausible structure to be made in regions of unobserved space, without the need for a feature vector to be computed for each position in the unknown region.

It is these new opportunities afforded by structured machine learning that drive us to pursue option number 3 in the next chapter.

Chapter 5

Structured completions of 3D scenes

In the previous chapter we demonstrated that we can learn a mapping from a depth image to an estimation of voxel occupancy in unobserved regions of space. We established the issues associated with predictions of geometry, in particular that it is the surfaces visible in the depth image which provide information about the occupancy of unobserved regions, but it is not clear for a single unknown voxel where to look to find the informative surfaces. We introduced two test datasets, and we were able to train and test on these datasets to make fairly accurate predictions of geometry. However, we noted that because we were making predictions on a *per-voxel* basis, the final predictions were often noisy and lacked the structure of real-world scenes.

Our previous occupancy model did not reward solutions which were spatially consistent. In order to make final predictions which maintain spatial consistency, in this chapter we present a method to locally map from a single point on a depth image to predictions of the occupancy value of multiple nearby voxels simultaneously. Here, we take an alternative approach to the previous chapter: instead of starting with each unknown voxel and inferring its state, we start with the observed image and use this to work out what the state of the unknown voxels are. We achieve our depth point to occupancy prediction mapping using *structured learning*.

Our structured learning model helps to enforce spatial consistency in the final out-

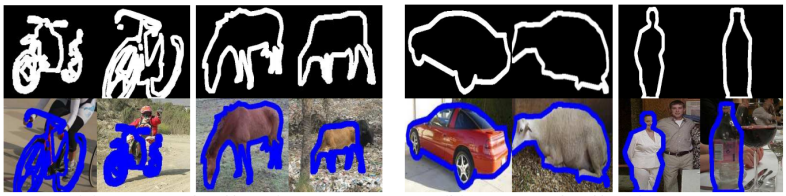


Figure 5.1: Examples of the *shape sharing* algorithm for image segmentation[93]. The intuition is that object shape can transcend class boundaries, so dissimilar objects can share the same shape as seen in (b). Of course, objects with similar class can also share similar shape, as shown in (a). Image adapted from [93].

put by making multi-dimensional predictions, in contrast to the unidimensional outputs in Chapter 4. Each multi-dimensional prediction will typically contain a small piece of real-world, observed training data. This ensures that each structured prediction made reflects not just a likely single label, but also a likely spatial configuration of labels in the region over which the prediction is made. In the broadest sense, there are many forms of structured learning, and some of the most popular have been where the solution is found from the final output space e.g. using an undirected graphical model (see [119] for an overview). In this chapter we follow previous work such as [39] which have made structured *predictions*. Here, each single prediction from a model is multidimensional, and naturally reflects the structure of the real world. We use this concept to develop a novel method for predicting unobserved geometry with structure and regularity 'built in'.

We take inspiration from recent work that segments objects from 2D images using silhouettes learned from different object classes [93] — see Figure 5.1. Their work showed that shape can transcend class categories, enabling shape predictions to be made without the need for semantic understanding. Because we care about shape, independently of semantic understanding, we are free to use training objects that differ from the objects present in the test scene.

There are two key contributions in this chapter that underpin this novel depth image to voxel geometry framework:

- ⇛ *Voxlets*: We use a structured Random Forest to learn a mapping from a point in a depth image to a structured prediction of the geometry in the region around that point. This is the first such application of structured learning to voxel predictions. We term each prediction of multi-voxel geometry a *voxlet*.

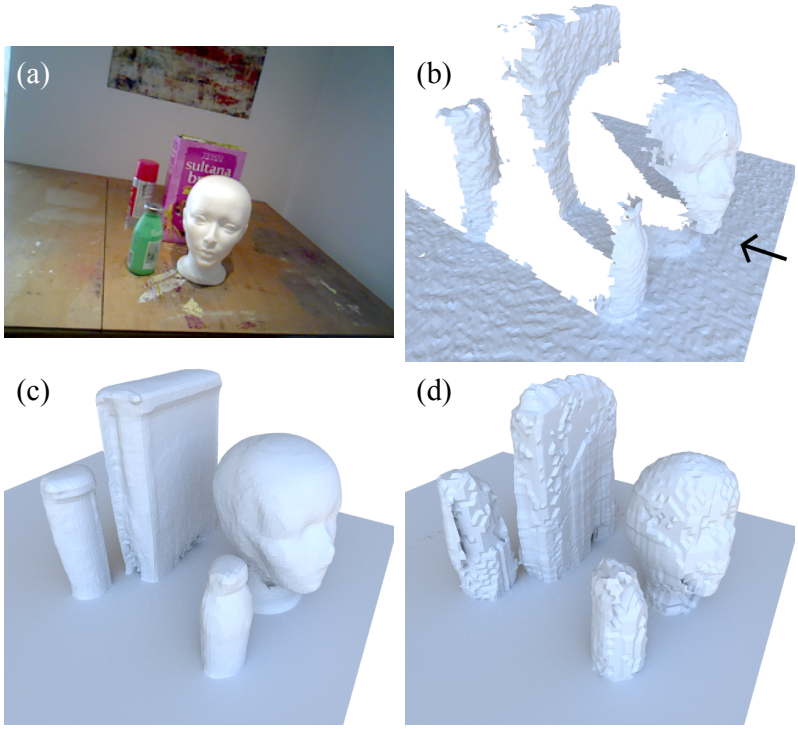


Figure 5.2: An example completion using our structured prediction algorithm. (a) Intensity image, for illustration only. (b) (Input) 3D projection of the depth image, captured from the black arrow's perspective, where occlusions induce large empty spaces. (c) Ground truth occupancy. (d) (Output) Our algorithm predicts a plausible completion of the tabletop objects' geometry.

- ⇛ *Fitted predictions*: As opposed to naively placing structured predictions directly into a scene, we demonstrate the efficacy of selecting the best predictions based on their fit to the observed geometry.

An example result of the structured predictions made in this chapter is displayed in Figure 5.2.

5.1 Related work

In Section 4.2 we reviewed previous works which aimed to make single-pixel or single-voxel predictions in images and grids. However, per-voxel (or per-pixel) predictions can suffer from noisy outputs, as we discovered in the previous chapter. To mitigate against these problems, *structured learning* has become a popular method for some computer vision tasks. As with standard supervised learning, structured learning finds a mapping from a feature space to a label space. In contrast to a traditional one-dimensional label space, however, a structured label space is *multidimensional*. The feature space in structured

learning problems is typically a local descriptor of the image, while the label space may be surface normals [53], human poses [13], image edges [40] or semantic labels [95]. This family of works provides inspiration for our approach.

There are many different ways of finding the mapping from feature to structured label space. For example, [13] cluster human poses, while [53] use an SVM-like formulation to find primitives which are both discriminative in feature space and informative in label space. We make use of Random Forests [16] for this machine learning problem. Originally proposed for regression and then single-label classification problems, such as our work in Chapter 4, they have since been adapted to make structured predictions for tasks such as semantic labelling [95] and edge detection [39]. In general, structured predictions can be faster and more regularized than a single dimensional predictor.

Unlike these previous works on structured prediction, we are making predictions into *unknown regions of the scene*. This adds a key challenge beyond the standard structured prediction paradigm. In our case, deciding *where* to make structured predictions becomes important. We want to make predictions where there is enough visible data to make an accurate prediction, but equally where there are enough unobserved voxels for the prediction to be useful.

Patch-based image completion This concept of making predictions in unknown regions bears some relation to patch-based image completion. Works such as [69, 29] typically use region-based data-driven approaches as it is very difficult to form true generative models over image appearance. For example, Hays et al. [69] look up possible completion regions in a large database of similar images, while Criminisi et al. [29] in-paint by selecting and combining multiple plausible patches from other regions of the input image. This differs fundamentally from our task, as image completion typically aims for a visually plausible output, irrespective of the accuracy compared to ground truth. Furthermore, our structured predictions are in 3D rather than 2D space. Deciding where to place each prediction adds a new degree of complexity above 2D prediction, and requires a novel solution. Finally, our high dimensional output space puts limitations on the direct application of such algorithms, and we therefore have to carefully design our algorithm to be tractable.

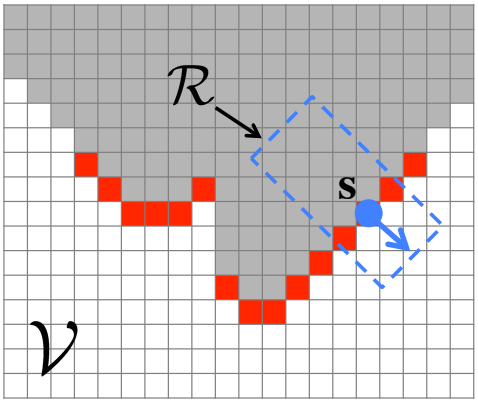
5.2 Overview of our structured prediction algorithm

As in previous chapters we model the geometry of the scene as a regular grid of voxels $\mathbf{V}$, where each $v \in \mathbf{V}$ represents the distance from the location of that voxel to the nearest surface as a truncated signed distance function (TSDF) — see Section 2.1 for details. We make the observation that an observed region on the depth image can be very informative about nearby 3D geometry completion. Our algorithm in this chapter maps a point $\mathbf{s}$ from the observed depth image $\mathbf{D}$ to a prediction of the TSDF in a voxel neighbourhood about that point, and the aggregation of such predictions for multiple points on the input gives our final TSDF prediction for the scene. A 2D overview of this approach is depicted in Figure 5.3.

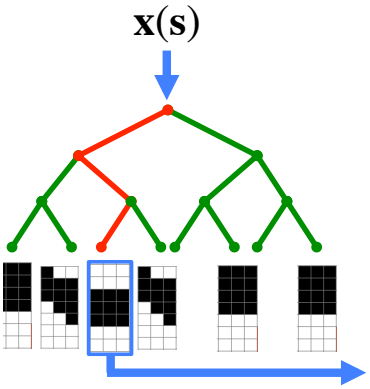
Support regions The support region $\mathcal{R} \subset \mathbf{V}$ is a set of voxels in the neighbourhood of $\mathbf{s}$, for which our model can make a prediction of the TSDF. Each $\mathcal{R}$ is a fixed-size cuboid of voxels, whose x-axis is aligned with the measured normal direction at $\mathbf{s}$ (Figure 5.3(a)). The size of $\mathcal{R}$ is defined so that it is large enough to capture local occupancy information at an object level, but not so large that it would span the entire scene. In a 2D world, the location of $\mathbf{s}$ and the direction of its normal can unambiguously define the location and orientation of $\mathcal{R}$. In 3D, however, there is an unconstrained degree of freedom, namely rotation of the cuboid about the axis of the normal. We resolve this by aligning the cuboid such that its z direction is coincident with the world z-axis, i.e. the 'up' direction of the scene, which we denote as $\mathbf{u}$. The top and bottom face of each cuboid region $\mathcal{R}$ is therefore parallel with the world's ground plane. To be precise, the local coordinate system of the voxlet at point $\mathbf{s}$ with normal $\mathbf{s_n}$ is:

$$R = \begin{pmatrix} -(\mathbf{s_n} \times \mathbf{u})^T \\ -((\mathbf{s_n} \times \mathbf{u}) \times \mathbf{u})^T \\ \mathbf{u}^T \end{pmatrix} \in \mathbb{SO}_3. \tag{5.1}$$

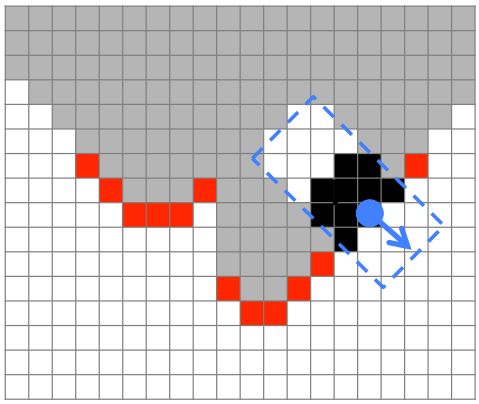
Voxlets At test time, we extract a feature description for $\mathcal{R}$ from the observed geometry at point $\mathbf{s}$. Building a structured Random Forest [40], we can make a prediction of the full, occluded, geometry inside of $\mathcal{R}$. We call this prediction of geometry a *voxlet*. The



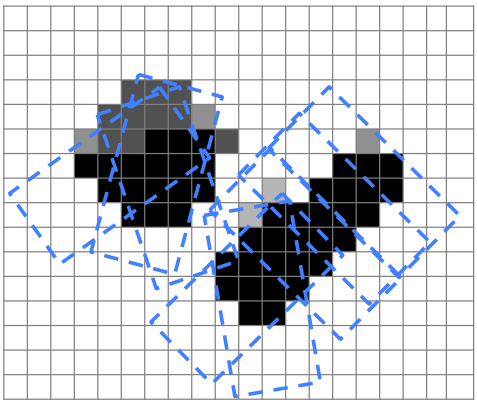
(a) Here, we show an overhead view of a scene which has been captured by a depth sensor. There is a (grey) region of unknown occupancy extending beyond the depth surface. At test time, we define a cuboid region of voxels, $\mathcal{R}$, around each query point, $\mathbf{s}$, aligned with the normal at $\mathbf{s}$.



(b) The structured Random Forest makes a prediction for the signed distance of each of the voxels in $\mathcal{R}$ given a feature $\mathbf{x}(\mathbf{s})$ computed from the observed geometry.



(c) This prediction is placed into the scene, and used to update the values of the voxels.



(d) The aggregation of multiple such predictions forms our final occupancy estimate.

Figure 5.3: A 2D overview of the structured prediction algorithm. Here we show the predictions as being binary for ease of illustration. In fact, we maintain a TSDF version of each voxlet and the final prediction grid accumulates these TSDF values.

voxlet, which comes out of the forest in canonical alignment, is then transformed from its local coordinate system into world space to fill the voxels in $\mathcal{R}$ (Figure 5.3(c)). The accumulation of multiple such predictions forms our final prediction of the full TSDF (Figure 5.3(d)).

5.3 Learning a mapping from features to voxlets

We pose unobserved geometry estimation, given partial observed information, as a supervised learning problem. More specifically, our goal is to learn a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that maps a feature vector $\mathbf{x} \in \mathcal{X}$, computed from observed geometry, to the output space $\mathbf{Y} \in \mathcal{Y}$ representing the corresponding 3D geometry in the region $\mathcal{R}$ around $\mathbf{s}$. Unlike standard classification, where the goal is to predict a category label for each $\mathbf{x}$, our output space is a three-dimensional array $\mathbf{Y} \in \mathbb{R}^{w \times d \times h}$ that encodes the TSDF values in the local region. The dimensionality of $\mathbf{Y}$ is prohibitively large, making it difficult to use standard multivariate regression approaches, e.g. [30]. Inspired by the recent work of Dollár and Zitnick [40], we use a structured Random Forest to learn the function f.

5.3.1 Training

Our training set, $\{(\mathbf{x}_1, \mathbf{Y}_1), ..., (\mathbf{x}_n, \mathbf{Y}_n)\}$, comprises region and feature pairs sampled from the full 3D reconstructions of scene. To train the structured forest we pass a random subset of 50% of the training set to each tree, starting at the root node. This use of a random subset for each tree is known as 'bagging'. It helps to reduce overfitting [16] and helps ensure that the predictions from each tree exhibit diversity. Each node is then tasked with splitting the data using the $\mathbf{x}$ variables such that the data sent sent to each child node are as similar as possible in shape, i.e. with similar $\mathbf{Y}$ values. One way of achieving this would be to find a split in the data which minimises the sum of squared differences of the set of labels at the left node ($\mathcal{Y}_L$) and right node ($\mathcal{Y}_R$), i.e. which minimises:

$$E(\mathcal{S}) = \sum_{d \in \{L,R\}} \sum_{\mathbf{Y} \in \mathcal{Y}_d} ||\mathbf{Y} - \bar{\mathcal{Y}}_d||_2^2, \quad \text{where} \tag{5.2}$$

$$\bar{\mathcal{Y}}_d = \frac{1}{|\mathcal{Y}_d|} \sum_{\mathbf{Y} \in \mathcal{Y}_d} \mathbf{Y}. \tag{5.3}$$

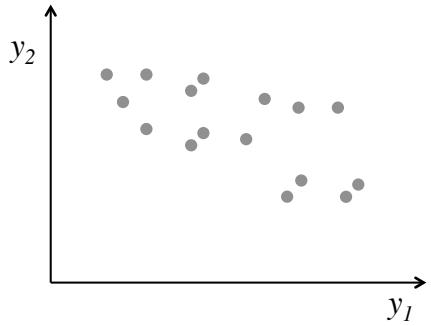
In effect this energy function rewards having a small spread of labels at the child nodes.

However, for our high-dimensional label space computing this energy change for each candidate split at each node in each tree would become prohibitively expensive. Instead of minimizing this loss directly, we follow [40] in approximating this loss at each node using a *classification loss*. To use a classification loss, at each node each $\mathbf{Y} \in \mathcal{Y}$ is assigned a proxy label $\in \{0, 1\}$. A split is then found in the data which minimises the classification loss, as if we were performing binary classification at the node. To convert the structured problem into a classification one, we create our two proxy classes by clustering: Before splitting the data at a node, we sample a different random subset of the dimensions of each $\mathbf{Y}_i$, reduce their dimensionality to M dimensions, and then cluster. Then a standard classification loss can be used on this new discretization, to evaluate the quality of different candidate splits for each $\mathbf{x}_i$ — in our case we use the Gini impurity measure. In practice, we efficiently perform this dimensionality reduction and clustering at each node using randomized PCA [65]. A training example is then assigned to one of the two possible clusters based on the sign of the value of its first principal component. See Figure 5.4 for a pictorial overview of this process.

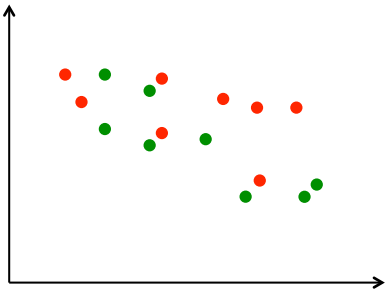
This process is repeated until we reach our maximum depth (which we set to 14), or we have fewer than five training examples at a node. In either of these cases, the node automatically becomes a leaf node, and splitting stops. Finally, as in [40], each leaf node stores the medoid of all the examples that have arrived there, which we refer to as a *voxlet*. We store the medoid for efficiency reasons but it is also possible to store multiple modes, e.g. [59].

5.3.2 Features

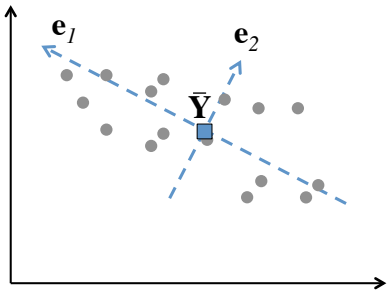
To describe the neighbourhood of $\mathbf{s}$ we use the surface feature $\mathbf{x}_{\texttt{surface}}$ as described in Section 4.5.3. The surface feature is suitable for our purpose as it describes the shape of the surface in the vicinity of the query point. It is also very quick to compute, and invariant to camera translation along the z-axis. By training on scenes captured from multiple angles we cover a wide range of possible camera $x - y$ translations and camera rotations.



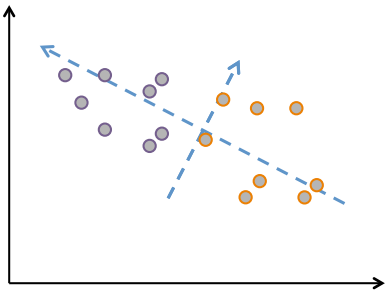
(a) Here we show the **label space** for a structured labelling problem. Each training data point at a node, shown here as a circle, is also associated with a point in feature space, which is not shown here. The aim is to find a split in feature space which corresponds to a 'good' division in label space.



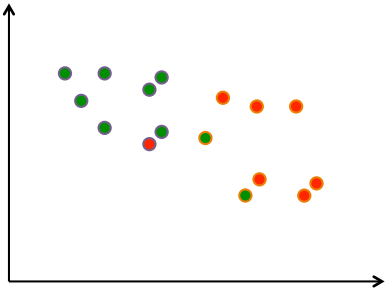
(b) Each proposed split in feature space divides the training examples in two; here we depict the *label space* result of a split, using red and green to colour the points. Traditionally, the split which minimises Equation 5.3 (i.e. which forms the tightest clusters in label space) may be selected from the set of candidates. This evaluation is expensive when dimensionality is high.



(c) In our work, following [40], we perform an initial classification in label space. We first find the principal directions $(\mathbf{e}_1, \mathbf{e}_2)$ in label space. These form a coordinate system centred on the mean of all the points $\bar{\mathbf{Y}}$.



(d) By looking at the sign of the first principal component of each data point, we can assign it a temporary, proxy label. We depict these proxy labels here as orange and purple outer rings.



(e) Finally we evaluate our splits, which again have been proposed in feature space (not shown here). We choose the split which assigns the points left and right (red and green) in a way that *most closely agrees with the proxy labels*. The split shown here has a fairly good agreement of split labels with proxy labels.

Figure 5.4: How the classification loss is used to make splits at nodes in the forest

Pre-segmentation for clutter Real world scenes contain clutter and interacting objects. This poses a challenge when we are extracting our training regions. While it is possible to represent the shape variation of isolated objects, modeling variations in arrangements of objects is much harder. This is intuitive, as the space of geometry induced by object combinations is much larger than that of individual objects. To overcome this problem, we perform an unsupervised segmentation of the *training* scenes, to separate individual objects. We use the same method for this segmentation of the training data as we do in Section 4.6.1, and an example results is shown in Figure 5.6(b). This segmentation encourages each training region to only model the shape of isolated objects. If objects are not well segmented at this stage, it is not a problem, but that node will be likely to make conjoined predictions at test time.

5.4 Predicting occupancy at test time

Each tree in our forest makes a prediction about how the volume surrounding a point in the input depth image is occupied. Our trees perform inference very efficiently, but in practice it is unnecessary to make a prediction densely for every location in the input, because closely neighbouring locations tend to yield similar predictions. We ignore locations where the normal points away from the camera, and also reject locations that point upwards (as defined by the scene's 'up' direction). This is because these upward-pointing locations are less *stable* and less *discriminative* than locations pointing sideways — the feature vector for each point on the top surface of a box, for example, is likely to be very similar. Specifically, we reject points whose normal has a z-component of greater than 0.98.

We then sample locations throughout the input image, spanning the spectrum of depths, to ensure uniform scene coverage. We only predict occupancy for regions about these locations (dots in Figure 5.6(c)). For this set of locations, we simply traverse a tree down to a leaf node, and return the voxlet stored there in training (Figure 5.3(b)). In Figure 5.5 we illustrate a few voxlets and their world positions, predicted for a real scene.

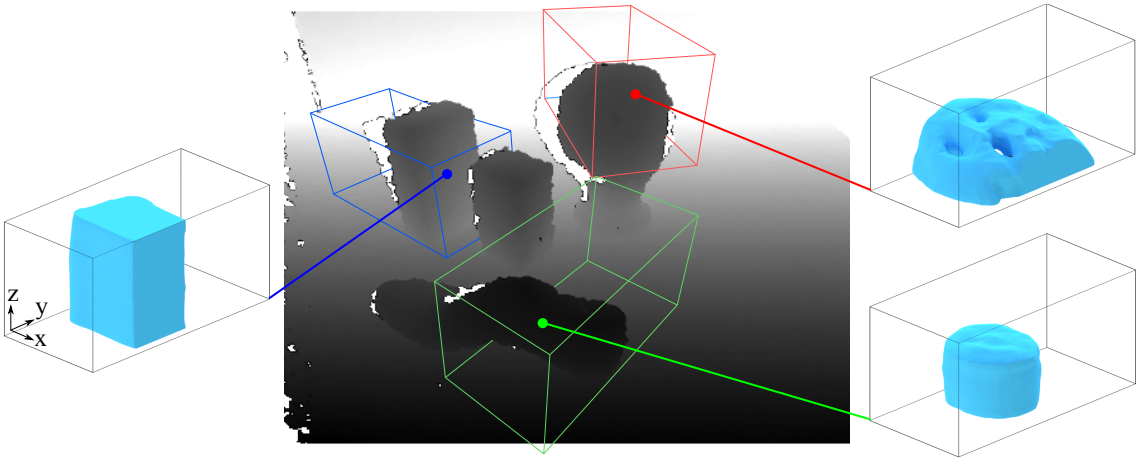


Figure 5.5: Each forest predicts the occupancy at each sample location, in the form of a voxlet, shown for the scene in Figure 5.6. Here we depict just three voxlets that have been meshed using marching cubes.

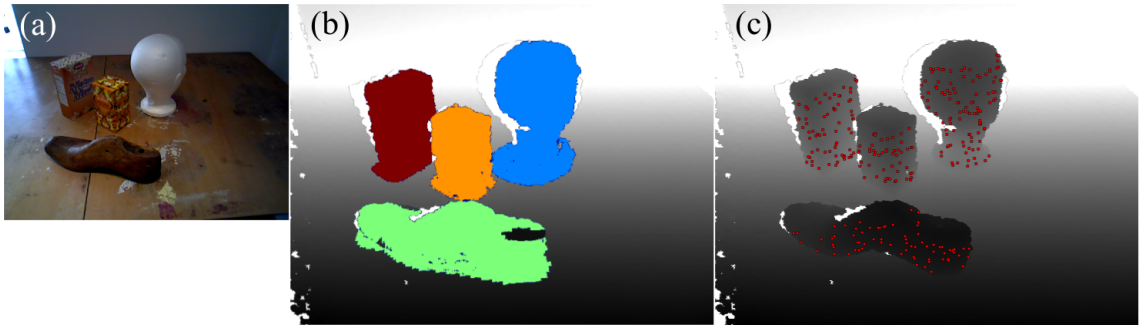


Figure 5.6: Pre-segmentation and sample locations for a single image, used in our prediction algorithm (a) RGB image, which is not used in our algorithm. (b) Automatic pre-segmentation of full geometry, performed if this were a training image. (c) Sample locations, shown as red dots, where occupancy *predictions* are made for a test depth image and where geometry is *extracted* from a training depth image.

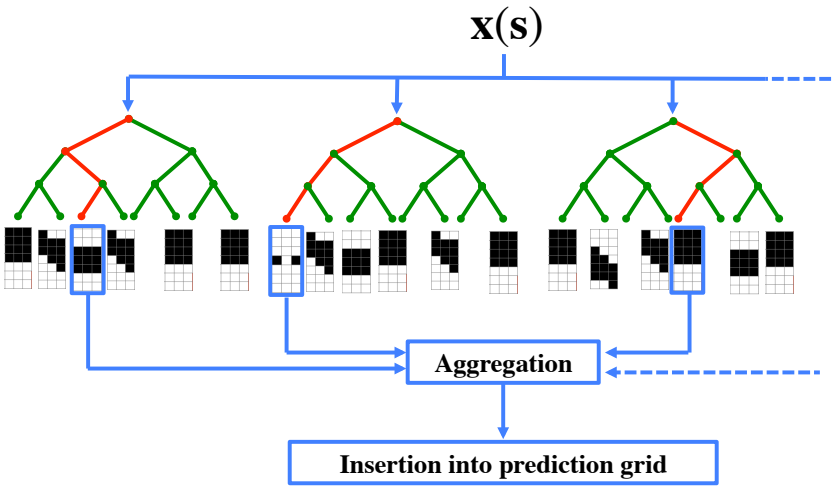


Figure 5.7: An overview of the pipeline for selecting from the tree predictions for a single query point **s**. The predictions for each tree are transformed into one single prediction using a method described in Section 5.4.1. The final, single prediction from the forest is then combined into the output grid, as described in section 5.4.2.

5.4.1 Choosing the best prediction from all the trees

Each leaf node in each of the n trees in our forest stores the medoid of the examples that landed at that node. For a given location in the input depth image, each tree will vote for a non-unique voxlet. We investigated three strategies to combine these region predictions from the different trees (Figure 5.7):

Forest Mean We simply take the mean of the voxlets as the forest prediction. We note that the truncation of the signed distance function helps to make this style of accumulation robust. A single incorrect estimation at a voxel can only be wrong by a maximum amount of $2d_{\max}$, where $d_{\max}$ is the level at which the distance function is truncated.

Forest Medoid The previous approach can produce artefacts as a result of the averaging. We propose selecting the medoid voxlet of all of the tree predictions (i.e. the medoid of the medoids) as an alternative approach to give more robustness to outliers.

Observed Fit Neither of the previous two approaches forces predicted voxlets to be consistent with the partially observed geometry from the input depth image **D**. To achieve this consistency, we first compute an observed TSDF for **D**. We then select the voxlet, from the set of n forest proposals, that best matches in the narrow band

of the observed TSDF, i.e. the voxlet which has the lowest sum of squared error across the voxels in the narrow band. We tried alternative distance measures, for example the sum of squared error across all the *visible* voxels (not just those in the narrow band). However, this gave worse results.

These alternatives are evaluated in Section 5.5.2.

5.4.2 Aggregating the predictions into a final TSDF

For the final prediction of the output TSDF grid $\mathbf{V}$, regardless of strategy, we average the predictions of the overlapping voxlets (Figure 5.3(d)). If a voxel in $\mathbf{V}$ has no predictions made for it, we mark it as empty (i.e. $d_{\max}$). For visualisation we use marching cubes [106] to convert the final predicted TSDF to a mesh by finding the level-set of zero.

5.4.3 Dimensionality reduction

Given the large dimensionality of ouptput space $\mathcal{Y}$, equal to the size of the support region $\mathcal{R}$, we perform an initial dimensionality reduction using PCA to 400 dimensions. Due to the large amount of redundancy in each region sample we empirically found this to have little impact on the quality of our results, yet it provides a large speed up at training time and reduces storage requirements. We evaluate the effect of this dimensionality reduction in Section 5.5.5.

5.4.4 Forest implementation details

We use an ensemble of $n = 40$ trees with simple axis aligned feature tests at each node, that are grown until there is a minimum of 5 examples at a node, up to a maximum of depth 14. When clustering the data at each node, we set the subset of random dimensions, M, for the randomized PCA to 20.

5.4.5 Sampling points for training data extraction and predictions

At test time, we only make predictions for 300 locations in the input depth image. As described above, we only sample points which are not pointing 'up' in the global scene sense, and which point 'towards' the camera.

5.4.6 Grounded and floating voxlets

We note that one limitation of only being able to make predictions in the region of a query point is that where occlusion is significant the rate of recall may be low, as each observed point is only able to make predictions in the neighbourhood about its 3D location. Figure 5.8(a-c) shows an example scene where occlusion causes a large area at the base of an object to have no predictions made for it at all. This problem can be significant in real-world clutter.

There are many options that could be used to mitgate this problem. For example, the location of each voxlet relative to the query point $\mathbf{s}$ could be additional dimensions in the label space, to be predicted by the forest. However, learning this type of spatial offset is non-trivial.

Instead we introduce a second form of voxlet in addition to the floating voxlet described in Section 5.2. This new type of voxlet we refer to as a *grounded* voxlet.

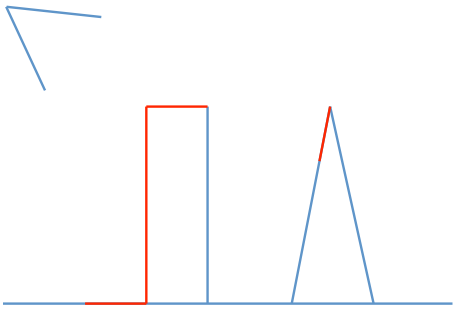
The difference between the grounded and the floating voxlet is that the floating voxlets have a z location aligned with the z location of $\mathbf{s}$, while the grounded voxlets all have a constant z location. The grounded voxlets are positioned such that their base is aligned with the ground plane of the scene. The rotation and $x - y$ location of grounded voxlets are equivalent to those for the floating voxlets, as described in Section 5.2. A pictorial overview of the use of grounded voxlets is shown in Figure 5.8.

In our experiments, we are able to predict one size of floating voxlet, and one size of grounded voxlet. This requires two different forests, each of which are trained separately. For a given sample location in a depth image at test time, we randomly choose one of the two forests to make a prediction.

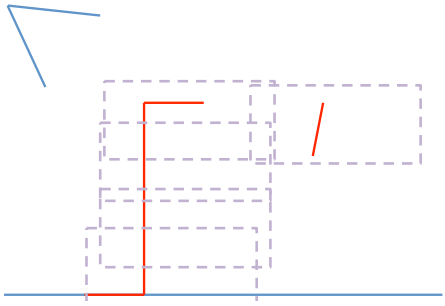
The floating voxlet, centred at $\mathbf{s}$, is longer in the y-direction ($15cm \times 30cm \times 15cm$). This is the direction that is approximately parallel to the normal at $\mathbf{s}$ (Figure 5.3(a)). This allows the voxlet to make a larger prediction *backwards* into the scene, compared to *sideways* which typically already has observed data.

The grounded voxlet, which has its base fixed to the ground plane, is taller ($15cm \times 30cm \times 37.5cm$). Diagrams of these voxlet sizes are shown in Figure 5.9.

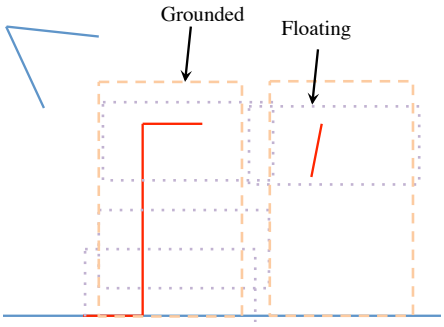
(a) An input scene consisting of two objects resting on a ground plane.



(b) The occlusion from the front cuboid means that from the position of the depth camera, only the top part of the cone is visible. (Visible surfaces shown in red)

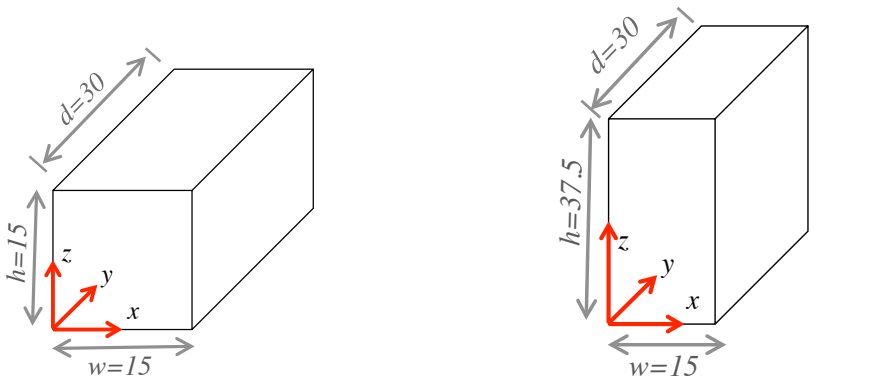


(c) Using just the floating voxlets (as described in Section 5.2), shown here in purple, we see that the base of the cone is not covered by any voxlets.



(d) We motivate the requirement for combining the 'floating' voxlet locations with locations *fixed* to the ground plane, shown here in orange. These voxlets ensure that predictions are made through the full height of the prediction space, ensuring objects like the cone have some prediction made for them at each height location.

Figure 5.8: A scenario used to motivate the requirement for both grounded and floating voxlets



(a) Floating voxlets (b) Grounded voxlets

Figure 5.9: The voxlet sizes and coordinate systems used in most of our experiments

5.5 Evaluation

We perform our evaluation of our algorithm in five parts. First, we demonstrate on isolated objects how our method overcomes the limitations of bounding boxes as a method for occupancy prediction. Next we compare against a state-of-the-art baseline on our two datasets, before then performing a qualitative assessment on some frames from the NYU dataset. We then perform experiments that give introspection into our algorithm, including showing some of the voxlets which are used most frequently, Finally we examine the effect of varying some of the parameters we have selected such as the size of the voxlets.

5.5.1 Isolated objects

First we evaluate our method on the BigBIRD single object turntable dataset [155], to show the limitations of a naive bounding box baseline, in addition to the method of Zheng et al. [178]. For the bounding box baseline we fit a minimum-area bounding box to the 3D points belonging to the object, as defined by the ground truth object segmentation mask provided by [155]. We are careful to remove points with normals perpendicular to the viewing direction, as these 'flying pixels' can have a large adverse effect on bounding box predictions. All voxels inside the bounding box are assumed to be occupied, while those outside are labelled as empty. Since we use the ground truth mask to fit the box, this can be viewed as the best possible bounding box given the observed data. For the comparison

Method	IoU	Precision	Recall
Bounding box	0.453	0.550	0.683
Zheng et al. [178]	0.365	0.579	0.466
Voxlets	**0.654**	**0.854**	**0.756**

Table 5.1: Evaluation of different completion methods on the BigBIRD dataset of [155]. The bounding box baseline uses ground truth object segmentation masks.

Method	IoU	Precision	Recall
Voxlets (Forest Mean)	0.236	0.941	0.240
Voxlets (Forest Medoid)	0.279	0.935	0.286
Voxlets (Observed Fit)	**0.864**	0.833	**0.737**

Table 5.2: A quantitative evaluation of the voxlets algorithm on the tabletop dataset. We compare against the baseline from [178] and the top results of the previous chapter

to Zheng et al. [178] we use the algorithm for reconstructing voxel occupancy as described in Section 4.2. We use their reference implementation, i.e. where their parameter $t = 3$.

The results from this quantitative analysis are shown in Table 5.1, with a subset of results pictured in Figure 5.10. The bounding box predictions suffer from poor recall. This is expected, as the method typically under-predicts the volume due to a lack of observed geometry. Our results successfully capture the overall geometry of each of the objects. The main drawback of our method is a smoothing effect, which affects some object edges.

5.5.2 An experiment comparing voxlet selection strategies

In this section we compare the three voxlet selection strategies proposed in Section 5.4.1. Table 5.2 shows a quantitative evaluation of the three alternative strategies for interpreting predictions from our structured forest. We can see that our 'Observed Fit' approach is better overall than other baselines, with very good recall and IoU. As a result of multiple conflicting overlapping predictions, 'Forest Medoid' and 'Forest Mean' tend to underpredict, resulting in higher precision but poorer recall and IoU (see Figure 5.11). We favour 'Observed Fit' because it chooses the prediction that agrees most with the observed geometry at each sample point, resulting in better completions. This is the voxlet selection strategy that we use for the remainder of our experiments.

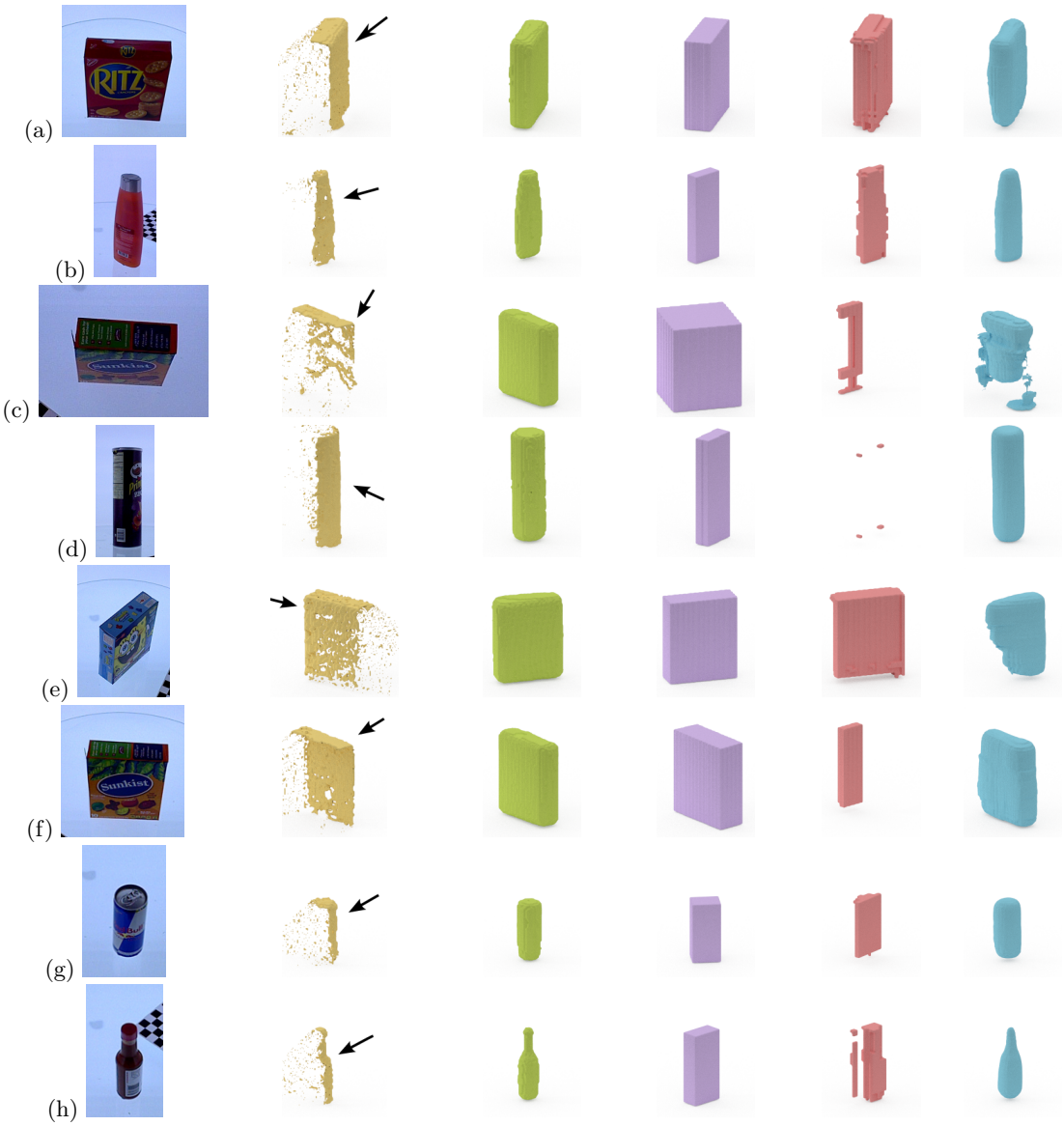


Figure 5.10: Comparison of different completion methods on the BigBIRD dataset of [155]. Here we show only two examples. The RGB view (not used) shows the depth camera's perspective, while rendered results shows side views for better inspection.

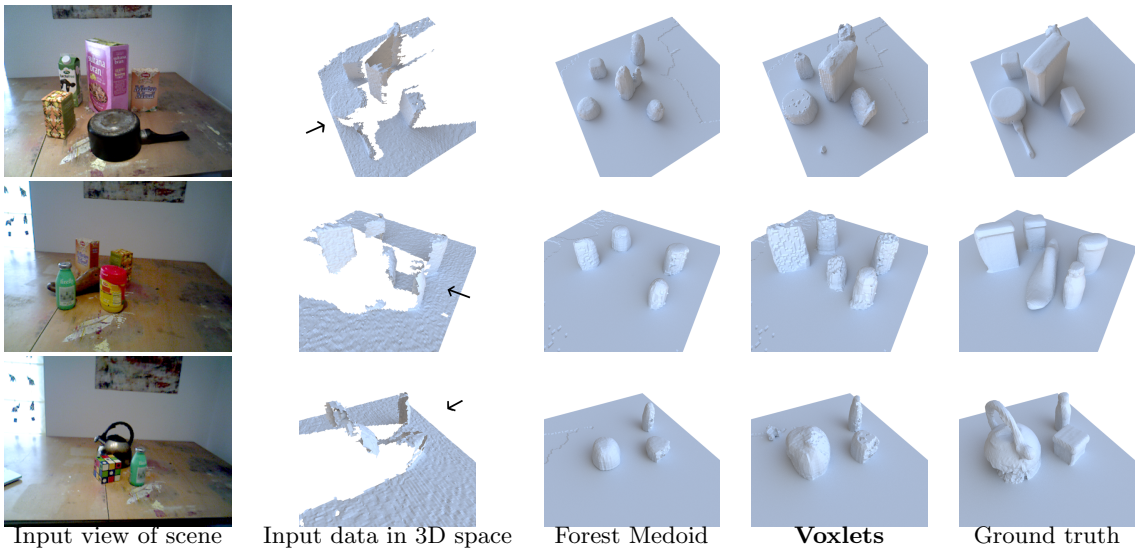


Figure 5.11: Qualitative comparison of voxlet selection strategies for our tabletop dataset. The input camera view is indicated by a black arrow. For clarity, we insert a ground plane during rendering and do not superimpose the observed input geometry on top of our predictions.

	Synthetic dataset			*Tabletop dataset*		
	IoU	**Precision**	**Recall**	**IoU**	**Precision**	**Recall**
Per-voxel prediction (Chapter 4)	**0.771**	0.906	0.831	0.627	0.823	**0.729**
[178] (t=2)	0.645	0.931	0.677	0.528	0.773	0.630
[178] (t=3)	0.322	**0.968**	0.326	0.378	**0.853**	0.405
Voxlets	0.737	0.864	**0.833**	**0.658**	0.811	0.717

Table 5.3: A quantitative evaluation of the voxlets algorithm. We compare against the baseline from [178] and the top results of the previous chapter

5.5.3 Experiments on our synthetic and tabletop datasets

To show how our algorithm performs on scenes with multiple objects in occluding configurations, we evaluate on the synthetic dataset and the tabletop dataset introduced in Section 4.3 and Section 4.4 respectively. To quantitatively evaluate our method we use the manually defined testing volumes for each dataset and make predictions within these regions, as in Chapter 4. The 'up' direction is found from the ground plane of this volume.

In Figure 5.12 we present qualitative results for the synthetic dataset. We can see that our algorithm is capable of reconstructing volumes even with large occlusions and limited input data.

Results from the tabletop dataset are shown in Figure 5.11, and an additional result is shown in Figure 5.2. Results are noteworthy because so many missing voxels in the input scan are correctly filled in, despite severe occlusions and fragmentation of objects.

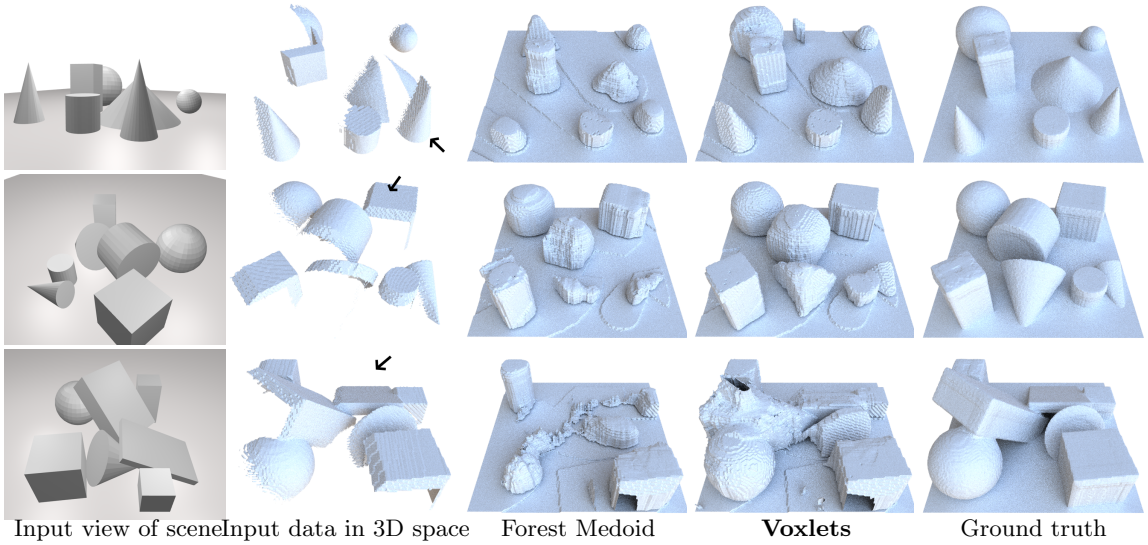


Figure 5.12: Qualitative results from our synthetic scene dataset. Each row shows a different configuration of synthetic objects. We note that, compared to Forest Medoid, our closest-matching method successfully recovers the shape of many objects. It is in areas of occlusions, and heavy clutter (row three) that are the most difficult to complete.

In Table 5.3 we show quantitative results for our voxlets method on both our datasets, comparing to the baseline of [178], and also the results from the previous chapter. We note that the method presented in this chapter is superior on the tabletop dataset, with the highest IoU and precision. However, on the synthetic dataset, the per-voxel prediction method gains a higher IoU. This gives an indication that the best algorithm for making predictions may be application specific.

The synthetic dataset consists of more dense clutter than the tabletop dataset, although with a more limited range of shapes. This may be one factor affecting the relative performance of the algorithms.

5.5.4 Qualitative scene results on the NYU dataset

An important question is the extent to which our models, trained on our datasets, will generalise to more diverse real-world scenarios. In Figure 5.13, we present results on depth images from NYU-Depth V2 [153], using our model from the tabletop training set. We assume the largest horizontal plane has been extracted from the scene, and the voxel grid is placed on top of that plane.

On the left, we are able to successfully recover the geometry of the scene, even with very sparse and noisy input depth. On the right, the input depth provides no cues as to

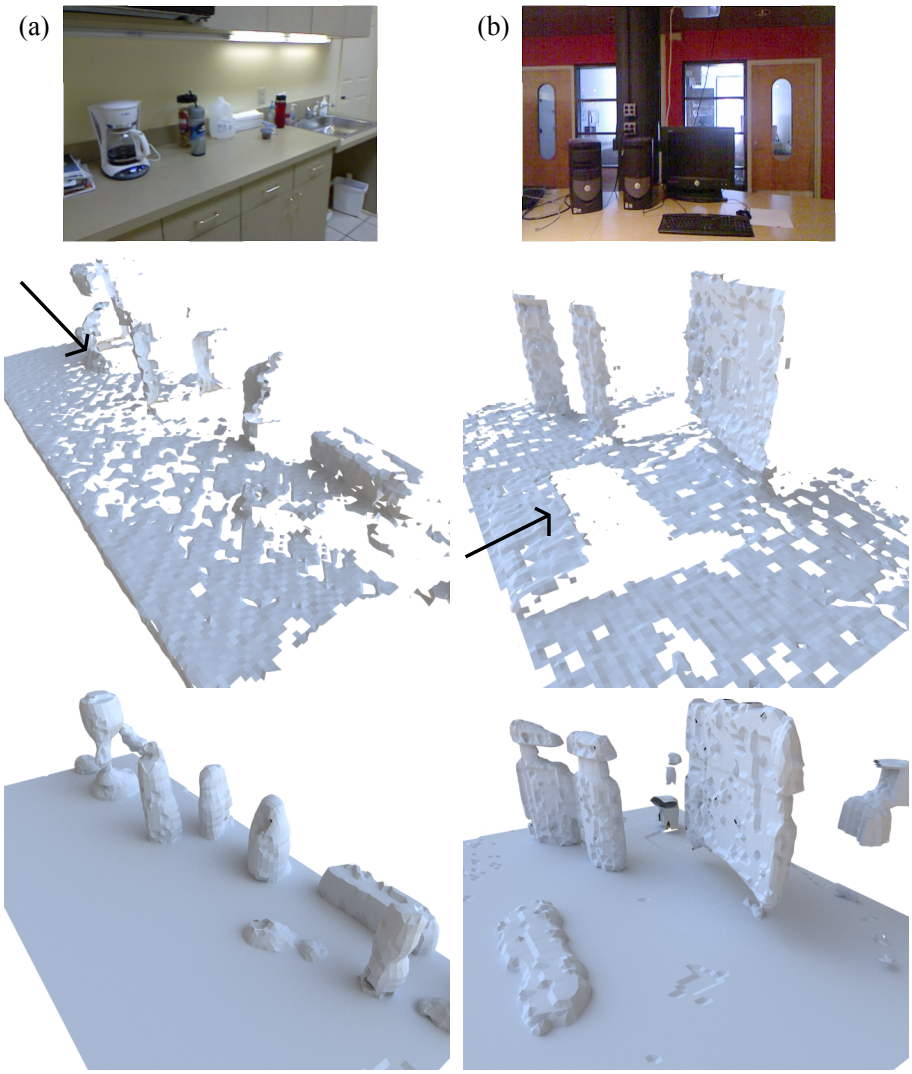


Figure 5.13: Occupancy predictions for two scenes from the NYU-Depth V2 dataset [153]

the shape of the desktop PCs, so the predictions are too shallow.

5.5.5 Using an oracle to gain insight into the algorithm

To get an insight into which parts of our algorithm are performing well, and which are causing failures to occur, we can perform experiments to give insight into the relative performance of different stages. We gain these insights by replacing various stages with an oracle that has access to the ground truth. We perform four such experiments:

`Oracle_gt`: With this model, instead of using the structured prediction, the ground truth voxels are extracted and then placed directly into the output grid. This baseline embodies the best output that a perfectly-trained version of our model could produce.

In particular, any failures from this oracle are likely to come about either due to quantisation in the voxlet space, or due to poor coverage of the voxlets in 3D space.

`Oracle_pca`: In this oracle the ground truth voxlets are compressed, then decompressed, using our pre-learned PCA model. This baseline evaluates how well the PCA model covers the range of voxlet shapes.

`Oracle_nn`: Here we use the ground truth data at each voxlet location to find a nearest neighbour training example to the ground truth in the training set. This is then used as the prediction at that location. Errors in this model are brought about by limitations in the span of the training set.

`Oracle_agg`: We use our structured Random Forest algorithm for prediction with the oracle at the aggregation step. Each voxlet is greedily added to the accumulator only if its inclusion increases the IoU score for the given scene. This oracle model demonstrates what score might be possible if we made careful choices about which voxlets to insert into the scene, instead of naively averaging all predictions together.

Results for these different configurations are presented in Table 5.4. These oracle-driven evaluations provide some key insights. Firstly, we notice that, on these scenes, the recall is lowered to 0.903 on the synthetic dataset just by the limitations in the coverage of the voxlets (`Oracle_gt`). This means that up to 10% of the ground truth voxels are simply not covered by the voxlet locations. Beyond this, though, the compression and reconstruction by PCA affects the scores very little (`Oracle_pca`). Using the nearest neighbour in the training set to the ground truth occupancy again hurts the recall on both dataset, but actually has very little impact on the precision (`Oracle_nn`). Finally, we can see that by using the predictions from the forest, but using our knowledge of the ground truth to decide whether or not to include them in the output grid, we can get almost as high a score as if we looked up the nearest neighbour to the ground truth (`Oracle_agg`). This suggests that the training data covers the space of voxlets reasonably well, but that making sensible decisions about which voxlets to include in the output grid could give a fairly large performance boost. We discuss one such option for achieving this in Section 5.6.1.

	Synthetic dataset			*Tabletop dataset*		
	IoU	**Precision**	**Recall**	**IoU**	**Precision**	**Recall**
`Oracle_gt`	**0.866**	**0.958**	**0.903**	**0.968**	**0.991**	**0.977**
`Oracle_pca`	0.864	0.954	0.898	0.920	0.979	0.939
`Oracle_nn`	0.815	**0.958**	0.845	0.759	0.937	0.790
`Oracle_agg`	0.815	0.939	0.860	0.712	0.877	0.815
Voxlets	0.737	0.864	0.833	0.658	0.811	0.717

Table 5.4: Quantitative evaluation of oracle-enhanced versions of the voxlets algorithm

5.5.6 Investigating the effect of voxlet size

For most of the experiments with voxlets we kept the size of our two types of voxlets fixed, as outlined in Section 5.4.6. Here, we vary the size of the voxlets used for training and testing on the tabletop dataset. We keep the ratio of dimensions the same for each experiment, so each voxlet is of size $(x \times 2x \times x)cm$. For this experiment we only train and predict using the floating voxlets — we do not use the tall, grounded voxlets at all. An interesting side-effect of this experiment is therefore to see the detrimental effect that just using one type of voxlet has on the results.

Figure 5.14 shows the effect that varying x has on performance on the tabletop dataset. It is clear that the voxlet size provides a trade-off between precision and recall, although the effect is not monotonic; instead, precision is highest with larger and smaller voxlets, while recall is highest with a mid-sized voxlet. We mark on the value of x we have used for our experiments with a dotted black line, and we observe that we have chosen a fairly sensible size for our experiments, with a good trade-off between precision and recall, maximising the IoU.

5.5.7 The most used voxlets from the tabletop dataset

Figure 5.15 shows the frequency of use of the 500 most used voxlets, when running the algorithm on the tabletop test set. This distribution includes both the short, 'floating' voxlets together with the taller voxlets which are fixed to the ground plane. While there is a bias towards certain voxlets, there is also a good spread of frequency across the top voxlets. The top 1925 voxlets are used 50% of the time.

Figure 5.16 shows a selection of the most and least used voxlets throughout the testing dataset. The different types of voxlet shown in each render ('tall' or 'floating') is noticeable

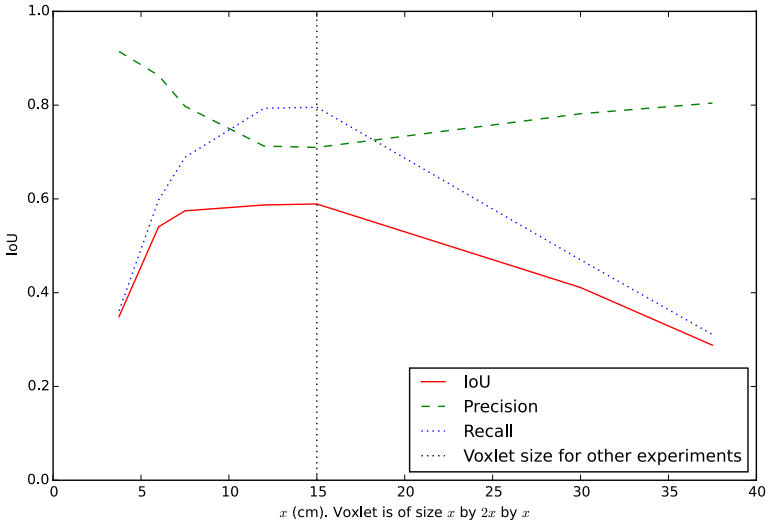


Figure 5.14: The effect of varying the voxlet size on the prediction performance. x is the size of the x and z dimensions of the floating voxlet; for this experiment we do not use the grounded voxlet at all.

by the varying sizes of the bounding boxes. Most of the most used voxlets resemble primitive shapes, such as cylinders and cuboids. This is expected, as we would expect these to be used in many locations in the test scenes. However, some of the top voxlets appear to be very specialised, e.g. those at rank 2 and 4.

5.5.8 Sensitivity to camera viewpoint

Following a similar experiment examining the effect of camera viewpoint in the previous chapter (Figure 4.21), here we examine how sensitive the voxlets algorithm is to camera viewpoint on a scene from the tabletop dataset. The results of this experiment are shown in Figure 5.17. Compared to the completions in the previous chapter, the voxlets algorithm is more sensitive to the viewing position of the camera. For this scene, approixmately half of the frames given an IoU of around 0.75, and the other half the IoU is closer to 0.5. Figure 5.17(b) shows a poor completion caused by missing data; here, the side of the large purple box has not been well observed, meaning that completions can not be made. Figure 5.17(d) shows an alternative failure mode. In this top-down view few points can be sampled for making proposals, so many objects are undercompleted. However, there are many successes. Figure 5.17(a), (c) and (e) are input views which are well completed,

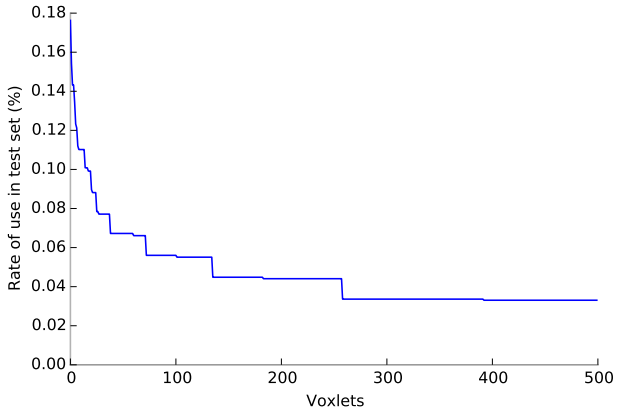


Figure 5.15: A distribution over the most popular voxlets in the testing dataset.

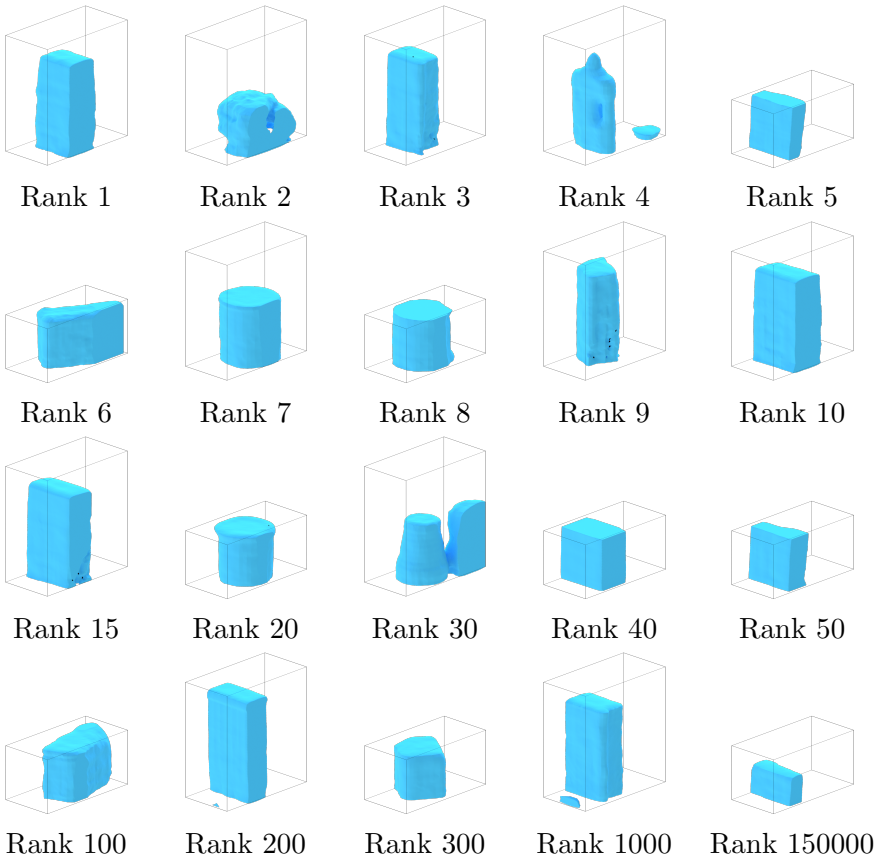
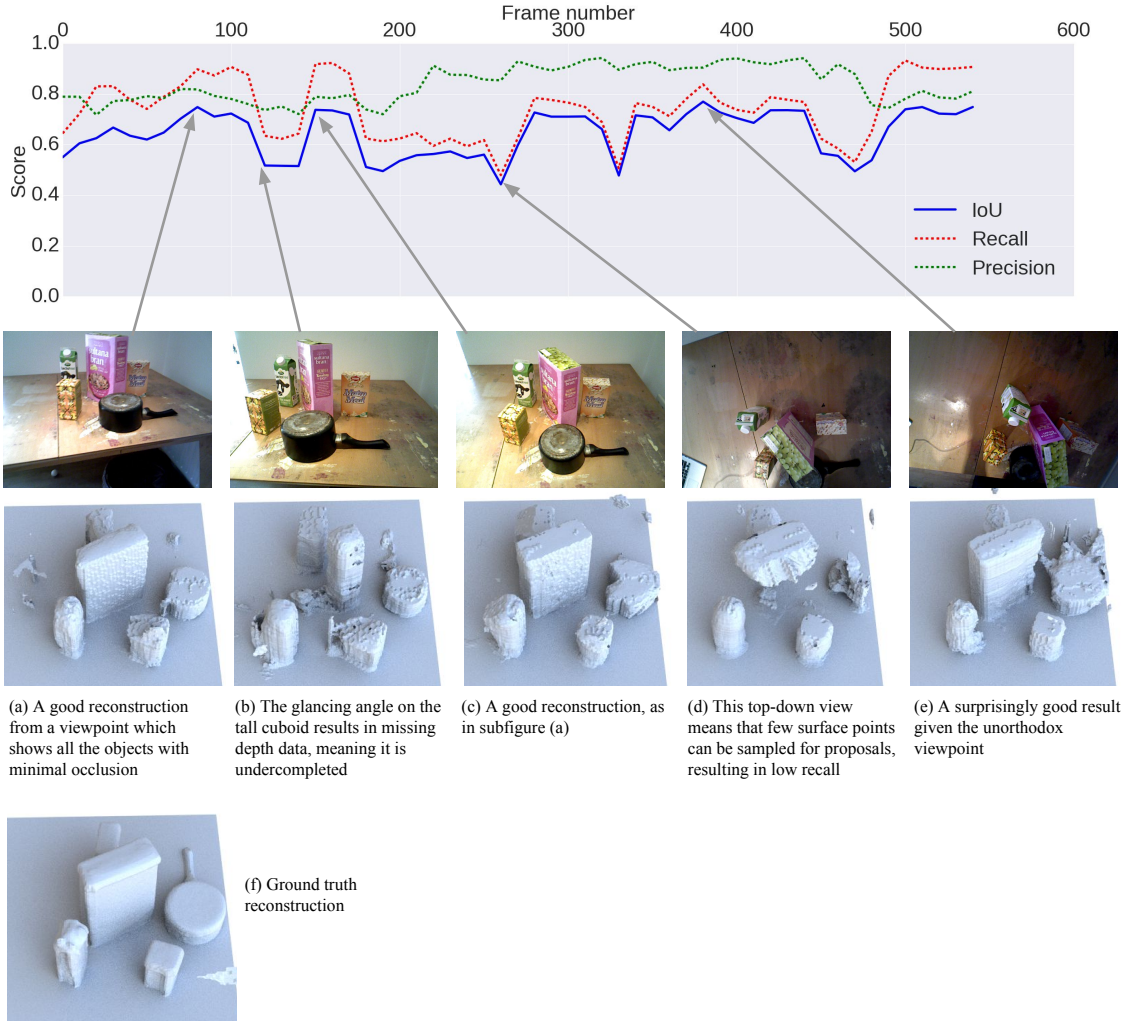


Figure 5.16: Some of the most and least popular voxlets, when predicting for the tabletop test set.



(a) A good reconstruction from a viewpoint which shows all the objects with minimal occlusion

(b) The glancing angle on the tall cuboid results in missing depth data, meaning it is undercompleted

(c) A good reconstruction, as in subfigure (a)

(d) This top-down view means that few surface points can be sampled for proposals, resulting in low recall

(e) A surprisingly good result given the unorthodox viewpoint

(f) Ground truth reconstruction

Figure 5.17: Sensitivity of the voxlets completion to changes in camera viewpoint, for a single video sequence from the tabletop dataset. (a)-(e) show completions from different camera viewpoints. Each completion is made independently, using only a single frame as input. The ground truth completion is shown in (f).

in spite of occlusions (a, e) and obscure viewpoints (e).

5.5.9 Per object performance

As in section 4.6.6 it is useful to examine which objects perform well and badly under this completion algorithm. In Table 5.5 we show the average scores for all the scenes containing each object. We can see that, surprisingly, the top scoring objects cover a wide range of shapes; a tea box, the kettle and the shoe. The worst performing item, the yellow brick, can be seen to be severely under-completed in the qualitative results. This is in contrast to the per-voxel prediction, where the brick was completed much more successfully (Table 5.5).

	Object	IoU	Precision	Recall
	Tea box	0.620	0.717	0.806
	Kettle	0.619	0.765	0.803
	Shoe	0.608	0.758	0.795
	Milk	0.600	0.749	0.815
		...		
	Red bottle	0.512	0.707	0.714
	Saucepan	0.511	0.810	0.651
	Fez	0.495	0.768	0.582
	Brick	0.455	0.806	0.499

Table 5.5: The four best and four worst performing objects with the voxlets algorithm, on the tabletop dataset. For each object, we give the average score of all the scenes which it appears in.

5.6 Conclusions and future work

We have demonstrated that we can successfully recover 3D geometry from only a single input depth image by making structured predictions. Our algorithm efficiently combines both shape selection and pose estimation, using simple feature test evaluations to predict local geometry occupancy. Voxlets are learned from training data, which has so far included only man-made table-top sized objects. Even similar objects vary in size and viewing direction, but objects from distinct semantic classes share enough 3D shape components to allow good, though not perfect, reconstructions.

Sharp edges are frequently rounded due to averaging. Also, as a supervised learning algorithm, our algorithm is limited by the data available at training time. This can be seen in Figure 5.13, as no computer-sized objects are in the training set. Currently, our voxlets are a fixed size, and success correlates with the test-scene having similar sized objects.

Timings and complexity Predicting the occupancy for a single scene takes less than 30 seconds using our unoptimized Python implementation. We found that memory was a limiting factor at both training and test time. This could be improved using a more efficient voxel storing scheme.

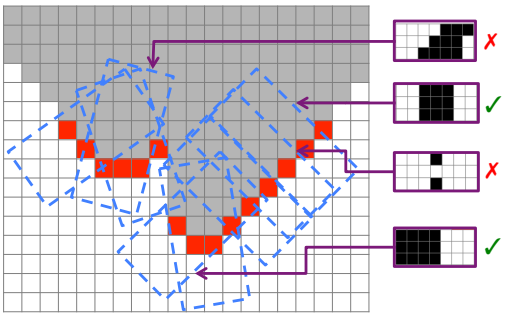
5.6.1 Future work

We present here two interesting opportunities for future research to improve the quality of the results from the algorithm

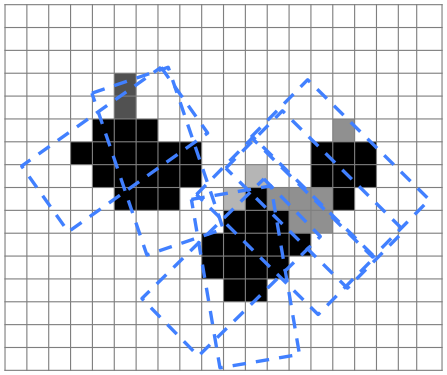
Physics-based reasoning For some applications, the quality of our predictions may already be enough, e.g. to aid robot grasping or navigation. However, there is currently nothing guaranteeing that our results are physically stable or smooth. *Learning* to make physically plausible predictions is difficult. How to best incorporate physics-based reasoning [178, 146] is still an open problem, but enforcing this strong prior may result in even more accurate results.

Trading off goodness-of-fit against coverage We showed in Section 5.5.5 that by selecting the best fitting voxlets from all the proposals, instead of accumulating all the

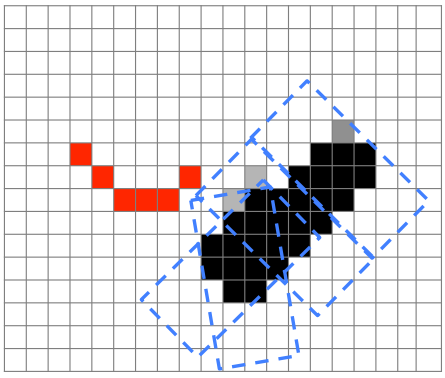
proposals from all the sampled points, we can get a better overall result. The intuition behind this is clear: at some of the sampled points, the forest is not able to provide a good prediction. In many cases the bad prediction from the forest is immediately obvious by virtue of the fact that the geometry of the predicted voxlet does not match the geometry of the scene at the proposed location. However, at other nearby points, the forest is able to make a good prediction which does match the geometry of the scene. By avoiding including bad-fitting voxlets into the scene at all, we should be able to increase the performance of the algorithm. However, the point at which we should stop adding in voxlets is unclear. If we make voxlet predictions *everywhere* in the target image, then the good predictions are marred and muddied by the voxlet proposals which are bad. Alternatively, if we add in *too few* voxlets, we end up failing to make predictions in many areas. We demonstrate this concept visually in Figure 5.18. Finding a way to balance 'goodness-of-fit' against coverage of the unknown space should be a key opportunity for future research.



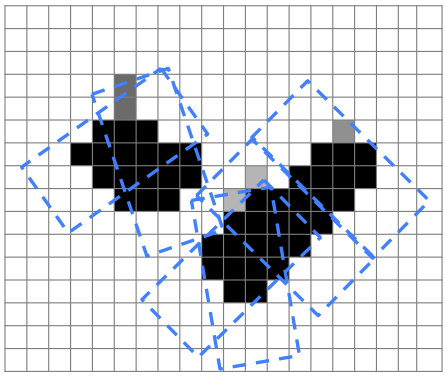
(a) Here we have made a set of predictions for the scene, as described in this chapter. Some of the predictions are good fits to the observed data (✓), while others do not match the observed data well (✗). This *goodness-of-fit* can be evaluated from the observed data.



(b) By averaging all the predictions together we find that the bad voxlet predictions (✗) can cause a poor final prediction.



(c) We could just average together the predictions which fit the observed data the best (✓). However, this may mean that some areas of the scene do not get any predictions made at all, leading to a low rate of recall.



(d) The best solution can be found by *balancing* goodness-of-fit against coverage. Here we have used well-fitting predictions (✓) where possible. Where no good predictions exist, we have made the best predictions possible in order to make some predictions for the unknown space.

Figure 5.18: An overview of proposed future work, regularising the voxlet predictions to ensure a balance between goodness-of-fit and coverage of unknown space.

Chapter 6

Learning to discover objects for object completion

In this thesis we set out to infer the occupancy of a scene given as input a single depth image. We have made some steps towards achieving this goal in Chapters 4 and 5. These methods used statistical inference, together with carefully constructed features, to predict the missing geometry of the scene.

However, the results we have obtained have only been *predictions* of the missing data. Our overall problem, as given, is an ambiguous inverse problem: The only real way to know what the back of the scene looks like is to observe it from more than one viewpoint. In Chapter 1 we described the problem that in many cases, it is impossible to capture all the images required for such a reconstruction. In this chapter, therefore, we introduce a novel method to recover the 3D shape of scenes.

If we had a library of training objects, each with ground truth occupancy, we might be able to recover the full 3D scene structure from a single depth image by fitting the 3D models to the scene. However, such a library of training objects is not available for many objects in the world (Section 3.3). We hypothesise that the scene viewed in many images is made up of individual objects, each of which we may have seen in other scenes. These other images may have been captured by a different device in a different location in the world, or by the same device nearby to the current image. If any of the other scenes containing the object has had its full geometry recovered (e.g. with a Kinect Fusion

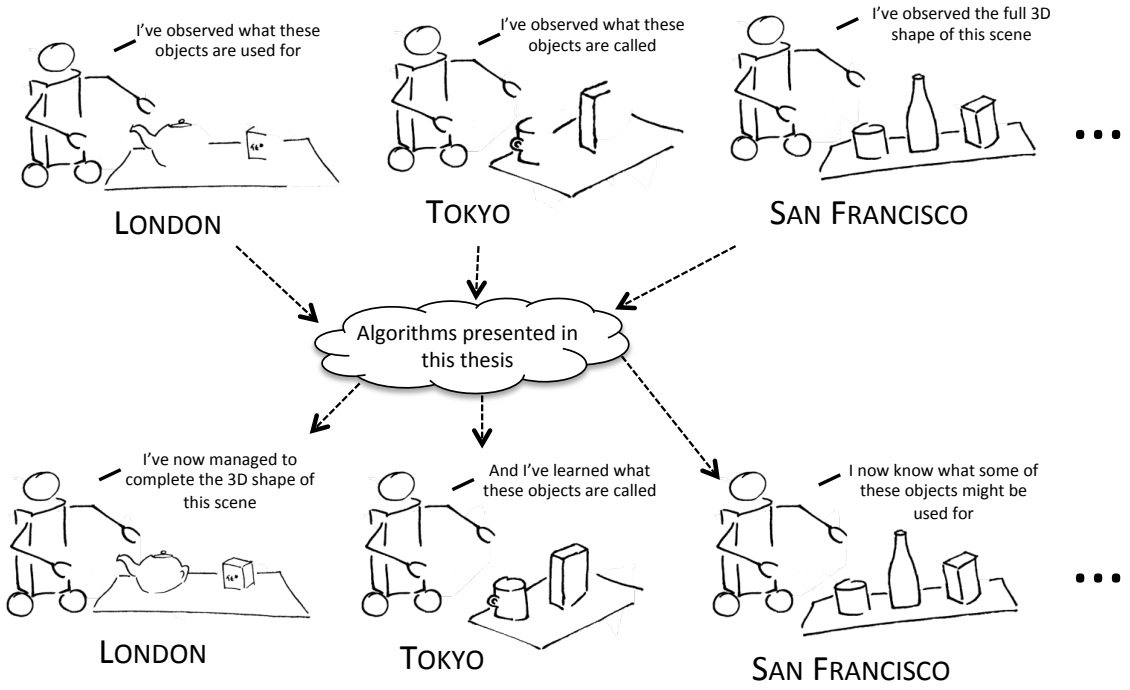


Figure 6.1: An overview image motivating the high-level concept of the algorithms presented in this chapter. By combining multiple partial views at an *object level*, we hypothesise that we should be able to complete many unknown regions of incomplete scans. Instead of using a library of training images, we require each object to be seen from differing viewpoints throughout a collection of input images.

system [80]), we can use the reconstructions of each *object* from these other scenes to help to directly complete the geometry of objects in the test scene.

However, this concept is non-trivial. Not only must a final alignment be found, at an object level, but regions of images corresponding to different objects must also be found between depth images captured of different scenes.

In this chapter we make two specific contributions to enable this method of 3D reconstruction:

1. We introduce a novel probabilistic framework for *discovering objects* from RGBD images. In contrast to previous works in this field, which typically require one or more human-set parameters for a clustering algorithm, we are able to *learn* how to discover objects by making use of training data.

2. We introduce a method for using these corresponding regions to enable the completion of some of the unseen parts of objects embedded in a test image.

6.1 Problem statement

We reason that many objects found in the world can be considered as instantiations of abstract object *templates*. Each template can be thought of as a mould which holds information on the three-dimensional shape and size of the object, its colour, texture, material and other physical properties. We define a *scene* as being a set of one or more object instantiations together with background environment (e.g. walls and floors), and further properties such as lighting and background colour. Each scene is then captured in a single static RGB-D image $\mathbf{D}$.

The aim of our *object discovery* system is to recover the number of object templates and each of their appearance models, given a set of input RGB-D images $\{\mathbf{D}_{1...N}\}$, and to find the set of pixels in each image corresponding to each object. We assume that we do not know information about the shape of object templates in advance, instead requiring this as an output from the algorithm. The recovered correspondences between regions in each image is then used as input to our system to recover missing parts of 3D shape of the objects in each scene. An overview of this approach is given in Figure 6.2.

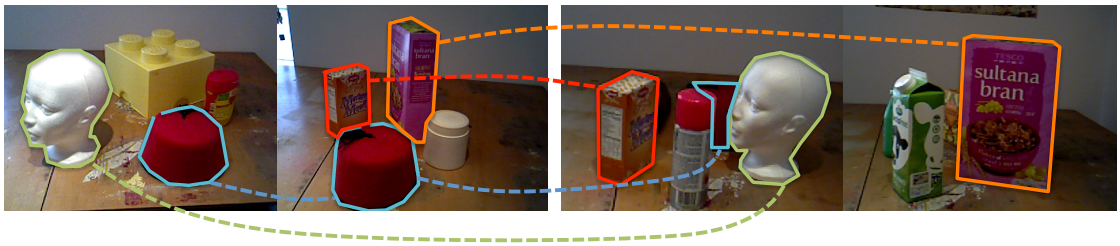
6.2 Related work

Three similar works to our system for shape recovery are [140], [169] and [94]. Vicente et al. [169] reconstruct the class-level 3D shape of objects seen in multiple RGB images from the PASCAL VOC dataset. Using a figure-ground segmentation and some sparse annotations to bootstrap, they then estimate the viewing direction of each input image and optimise over the visual hull to recover the shape. Our approach doesn't rely on such bootstrapping, and finds correspondences between objects within cluttered scenes to enable object shape recovery. Minkim et al. [94] use automatically-segmented training data to help form a full 3D reconstruction of their test-time scenes. Similar to our approach, they perform a training stage to capture geometry of objects, followed by a test phase where this geometry is applied to the test image. Ruhnke and Steder [140] combine point clouds of multiple views of each object in their scenes using ICP. Their output is point cloud reconstructions of individual objects and their pairwise score for discovering matching segments is based on attempted alignments between every pair of segments — this becomes very expensive

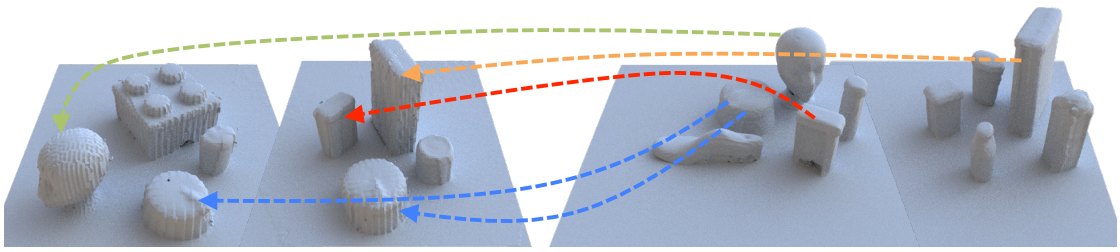
Partially observed *Fully observed*

(a) At test time, we assume that some scenes have had their full geometry captured (right) , while others have only been viewed from a single depth image (left).

(b) We can segment out individual objects from all the observed depth images.



(c) Using algorithms presented in this chapter, we can find correspondences between all the segmented regions.



Partially observed *Fully observed*

(d) Some of these clusters of regions include segments both from segmented images and partially-observed images. In this case we can use the geometry from the fully-complete objects to fill in the missing geometry in the partially-observed images.

Figure 6.2: An example of how we can use object discovery to complete missing geometry

as the dataset size increases.

Some works have used semantics to help improve localisation and mapping. Xiao et al. [176] use object labels to improve SfM reconstruction from RGBD data, while Salas-Moreno et al. [142] use detections of known objects to improve tracking and reconstruction. Alternatively, Herbst et al. [72] rely on objects having been moved between two observations of the same scene. They use 'scene differencing' to discover the shape and spatial extents of the moved objects.

6.2.1 Related work in object discovery

Object discovery and segmentation is a similar task to cosegmentation of images, where two or more images known to contain the same object are segmented into foreground and background regions. Cosegmentation is usually limited to segmenting just one object from two images, and works in this field usually focus on fast and accurate segmentation methods, rather than object discovery. A typical method for achieving this goal is to select pixels (or superpixels) from each image that maximise the similarity of the histograms between the foreground regions; this can be achieved efficiently through a modified graph-cuts algorithm [138]. Rubio et al. [139] verify cosegmentation matches by enforcing spatial neighbourhoods between matching regions, while allowing their global configuration to vary to enable the detection of deformable objects and changes in camera position. Similarly, Cho et al. [26] cluster matches between local features to hypothesise matching regions within image collections. This is used to form a confidence map over possible locations of repetition, which is then combined with the GrabCut segmentation algorithm [136] to achieve pixel-accurate segmentations. On single images, Carreira et al. [18] train a Random Forest regressor [16] on good, ground truth image segmentations. This is then used at test time to assess the probability of each of a pool of possible segments being a good object match within an image. Vicente et al. [170] extend this work to use pairwise matches between segments to find the best pair of segmentations, thus ensuring the found segments are well-segmented objects and which look like each other. However, their method is limited to discovering only one class of object at a time, and as with [181] they assume exactly one object instance per image. More recently, Fu et al. [55] perform cosegmentation on multiple RGBD images, using multiple segmentations to

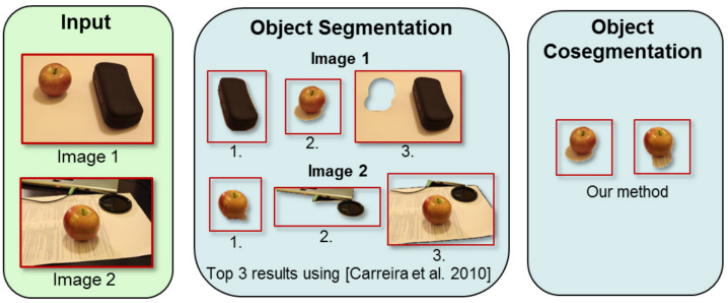


Figure 6.3: Object cosegmentation from Vicente et al. [170].

Author	Data	Segmentation method	Clustering method
Shin et al. [149]	Laser	Smoothness over normals	Branch-and-bound, pointwise alignment
Kang et al. [88]	RGB	Graph-based [46, 36]	Community discovery in networks [166]
Russell et al. [141]	RGB	Normalised cuts [148]	Latent Dirichlet Allocation [10]
Moosmann et al. [117]	Laser	Convexity constraints	Mean shift [56]
Ruhnke & Steder [140]	Laser	*Manual*	Spectral Graph Partitioning [120]
Endres et al. [44]	Laser	*Manual*	Latent Dirichlet Allocation [10]

Table 6.1: Algorithms used for object discovery in the 'segmentation → clustering' paradigm.

find accurate delineations of each object. All these cosegmentation works are limited to only discovering one class of object, while we will not know at test time how many classes of object we wish to discover.

Discovering *multiple* objects in a set of images is a 'chicken-and-egg' problem: it is hard to know what each object looks like until they are segmented, and it is hard to know how to segment out objects until we know what they look like.

When discovering textured, patterned or otherwise *feature-rich* objects or classes, the spatial arrangement of features on the objects can be exploited to find objects and their appearance simultaneously. Liu and Chen [105] use a traditional 'bag-of-words' topic modelling approaches to reward pairwise matches between image regions where patches form consistent spatial configurations, while Philbin and Sivic [124] formulate a stricter setup, where each object is modelled by a virtual 'pinboard' of feature locations; this naturally favours finding relatively planar objects with well-defined and adequately spaced visual features.

Most systems which have attempted some variant of object discovery, however, start by applying a generic segmentation algorithm to hypothesise candidate object locations in the images, before taking the second step of finding groups of matching regions:

Set of input data → Segment out regions corresponding to objects → Cluster matching regions

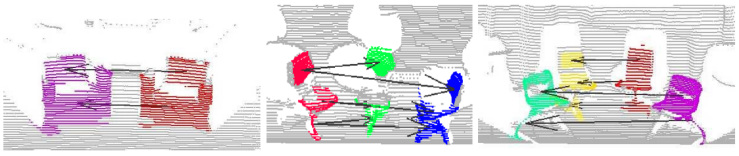


Figure 6.4: Example results from Shin et al. [149]. Lines indicate matches found between parts of objects.

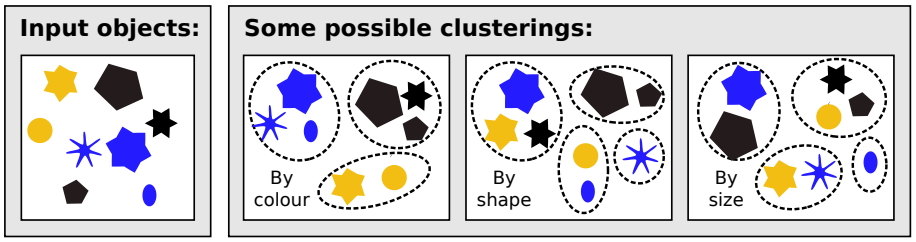


Figure 6.5: Looking at different properties can give very different clusterings.

Some interesting examples are shown in Table 6.1. All these object discovery algorithms work on either laser data or 2D images and on different types of objects—for example indoor scenes or outdoor environments. Clearly the type of segmentation and clustering algorithm used will be somewhat dependent on the data type and the type of objects being analysed. Russell et al. [141] was a key work which uses this method. Their 'soup of segments' is formed using normalised cuts [148], and segments are clustered using Latent Dirichlet Allocation [10].

6.2.2 Related work in clustering

As we have shown, the second stage of most object discovery algorithms, after hypothesising segmentations, is to reduce the more difficult scene understanding problem down to a clustering problem.

Clustering is a widely explored problem in statistics and computer vision. The goal of clustering is to assign a label y_i to each item $\mathcal{O}_i$ in a set of M items $\{\mathcal{O}_1, \ldots, \mathcal{O}_M\}$, such that that similar items have identical labels while dissimilar items have different labels [182]. We denote the set of all labels as $\mathcal{Y} = \{y_1, \cdots, y_M\}$.

This stated objective for clustering raises the obvious question of what we mean by the terms *similar* and *differing*. Looking at different aspects of objects' properties may give different clusterings (Figure 6.5), and for each property there are various degrees of 'similarity'. In fact, many clustering methods avoid ever formally defining these measures,

and instead rely on good input data and one or more user specified parameters in order to constrain the problem. These parameters may be the number of clusters required (e.g. k-means [108]) or an alternative parameter (e.g. mean-shift [56] requires a user-specified kernel function). For some applications it is suitable for a user to specify the number of clusters. However, for our situation this is not possible. Some typical clustering algorithms include:

k-means aims to minimise the objective function defined as the squared distance from each item to its respective cluster centroid. The number of clusters is user-specified. See e.g. [129] for details.

Spectral clustering [118], in which clustering is performed in a lower dimension subspace than the original data using the eigenvector decomposition of a similarity matrix between items. The number of clusters is typically user-specified.

Affinity propagation [54] operates by exchanging messages between points until a good set of exemplars and clusters is found. Its 'self-similarity' parameter controls the likelihood of an item becoming a cluster centre point.

Topic models are a family of techniques for discovering distributions of discrete 'topics' in collections of data, using a hand-defined generative model [171, 124]. Latent Dirichlet allocation is a popular topic model for clustering [10, 44]. Most topic modelling approaches therefore use just one feature type, often a histogram over local features—this has been shown to be sensitive to noise [50].

Automatic detection of the number of classes Some clustering methods have been developed to automatically estimate the number of classes in the data. x-means [123] is a modified version of k-means, which iteratively adds cluster centroids until a minimum value for Bayesian information criterion is reached. Like k-means, this algorithm is still sensitive to outliers. Hierarchical Dirichlet processes [161] is a topic modelling approach to cluster assignment which places a prior on both the number of and the assignment to classes, using the Chinese restaurant process. Philbin and Sivic [124] use a topic-models approach to object groupings, and exhaustively solve their model under a number of different values for K before choosing the model with the highest log likelihood.

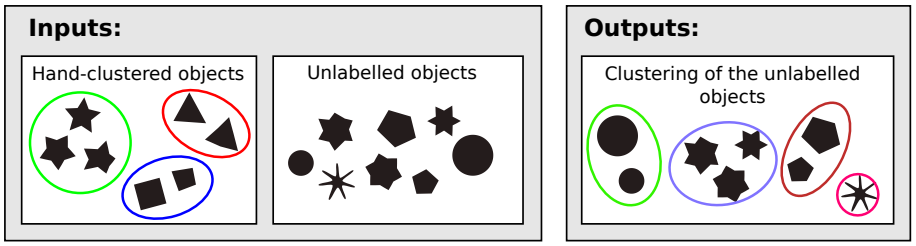


Figure 6.6: Inputs and outputs of a supervised clustering algorithm.

Distance measures in clustering Most clustering algorithms use a distance metric to measures degree of dissimilarity between data points, for example Euclidean distance or Mahalonobis distance. Distance measures only provide a measure of *dissimilarity*. In our work we don't just want to learn when objects are *dissimilar*, we want to also learn when they are *similar*, which measures in the range $[0, \infty]$ do not give any concept of.

In summary: Most object discovery, scene labelling and clustering algorithms rely on the user manually stating the number of types of objects expected to be found [181, 141, 171, 92]. In our work we present a novel object discovery method which uses training data to bypass this problem. This uses a form of supervised clustering (Figure 6.6).

6.3 Our object discovery methodology

In our work, we look to find groups of matching image regions, where each image region $\mathcal{R}$ is a subset of all the pixels in an RGBD image. Our algorithm for finding these groups of matching image regions consists of three steps:

1. We first hypothesise regions which potentially correspond to object locations, using a simple segmentation algorithm.

2. For each pair of segments we then compute a probability that they should be given the same label, taking into account both their similarity and the likelihood that each region is actually a good segmentation of an object.

3. Finally we regularise these pairwise and unary scores to group mutually consistent segments, to form our set of discovered objects.

We now explain and describe these steps in more detail.

(a) Input RGBD image (b) Planes removed (c) Spatial separation

Figure 6.7: Pictorial overview of our segmentation pipeline. Note that in the spatial separation step, objects which share the majority of their boundary with the image border are removed: see Section 6.3.1

6.3.1 Segmentation

We desire an object segmentation algorithm which will accurately delineate the boundaries of individual objects in each image $\mathbf{D}$, i.e. to propose a set of regions $\{\mathcal{R}_{1,\dots M}\}$, where $\mathcal{R}_i \subset \mathbf{D}$, such that each object's projection in the image space will have a large overlap with exactly one region. Many promising object segmentation algorithms have been proposed — for example [86, 114].

For our segmentation we use a very basic technique, which is suitable for the scenes in our evaluation data. First we detect and remove large planar regions from our scenes with the RANSAC implementation of [153]. To form complete object segmentations, we then use a simple top-down segmentation, placing pixels in the same region if they are neighbours in image space and the 3D spatial distance between them is less than a threshold t_d. For our data we use a value $t_d = 0.04$ m. Finally, regions for which the majority of their boundary is shared with the image border are removed. An example segmentation pipeline is shown in Figure 6.7.

This initial segmentation of our RGBD images is not a main focus of this work, and for more complex and cluttered scenes a different segmentation method will be required.

6.3.2 Pairwise measures between regions

Given a pair of image regions $(\mathcal{R}_i, \mathcal{R}_j)$, which may come from different images, we want to know the probability p_{ij} that the two regions should be placed in the same cluster, i.e. that they depict the same object. As many of the regions will not correspond to accurate delineations of objects (see Figure 6.7), we assert that p_{ij} is the probability that the two

Feature name	Type	Captures	Size
Bounding box aligned to principal components of points	Region	Size	3
Shape distribution [121]	Region	Shape	50
Histogram of gradients of RGB image [96]	Local	Texture	9
Mean RGB values [96]	Local	Colour	6
Spin image histogram	Local	Shape	50
Histogram over RGB values [96]	Local	Colour	64
		Total =	182

Table 6.2: The features used in our object discovery algorithm. The precise algorithms we use, and the parameters selected, are described in appendix C.

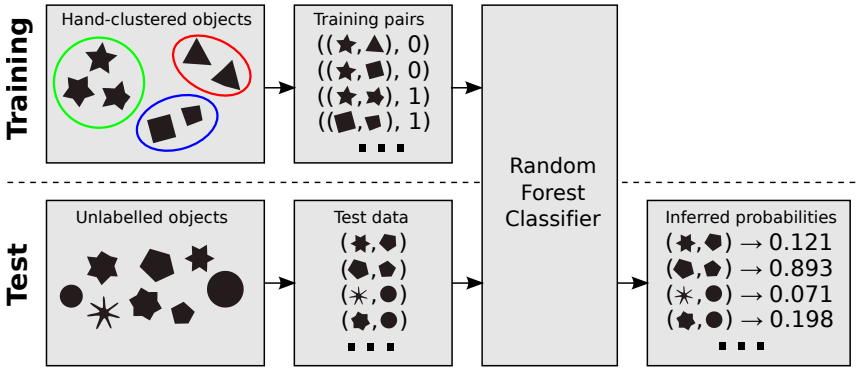


Figure 6.8: Overview of the weight assignment algorithm, used to find a probability of similarity $Pr(y_{ij} \mid \xi_i \wedge \xi_j)$ between each pair of regions.

regions depict the same item *and* that both regions represent valid objects:

$$p_{ij} = Pr(y_{ij} \wedge \xi_i \wedge \xi_j), \tag{6.1}$$

where ξ_i denotes the event that region $\mathcal{R}_i$ is a good segmentation of a single object in the scene (as opposed to background clutter or a part of one or more objects), while y_{ij} indicates the event that both regions depict the same item. We use the definition of conditional probability to reformulate Equation 6.1 like so:

$$Pr(y_{ij} \wedge \xi_i \wedge \xi_j) = Pr(y_{ij} \mid \xi_i \wedge \xi_j) Pr(\xi_i \wedge \xi_j) \tag{6.2}$$

We now describe how we compute each of the two quantities on the right hand side of Equation 6.2.

Computing pairwise scores $Pr(y_{ij} \mid \xi_i \wedge \xi_j)$

It can be difficult to directly compare two regions in their native 3D space, due to differing viewpoints, object pose and lighting in image regions. Instead we map each region to a feature vector $\mathbf{x}_i$, with the aim to remove some of the unwanted information contained in $\mathcal{R}_i$ (such as pose), while retaining the information relevant to our final objective (such as size and shape). The features we use are given in Table 6.2. The precise algorithms we use, and the parameters selected, are described in Appendix C.

We wish to find a way to represent each of our image regions such that similarities are retained, while differences induced by changing viewpoints and lighting conditions are ignored. Our pairwise approach requires a feature vector $\mathbf{x}_{ij}$ for each pair of regions $(\mathcal{R}_i, \mathcal{R}_j)$; this should capture similarities and differences between the objects. We define our pairwise feature vector as a component-wise absolute difference between the feature vectors of two regions:

$$\mathbf{x}_{ij} = |\mathbf{x}_i - \mathbf{x}_j| \tag{6.3}$$

$$\equiv (|x_{i_1} - x_{j_1}|, \cdots, |x_{i_n} - x_{j_n}|). \tag{6.4}$$

Each $\mathbf{x}_{ij}$ is therefore of the same dimension as $\mathbf{x}_i$; each element of $\mathbf{x}_{ij}$ captures an absolute difference in one feature dimension. This absolute difference function, also used in a similar way by [177], has the convenient properties of being symmetric and having a simple physical interpretation. Given each $\mathbf{x}_{ij}$, we then wish to estimate the probability that $\mathcal{R}_i$ and $\mathcal{R}_j$ represent views of the same object given that they both actually represent objects. We treat this as a classification problem, where the classes correspond to $\{y_{ij}|\xi_i \wedge \xi_j, \neg y_{ij}|\xi_i \wedge \xi_j\}$, where again $y_{ij} = [y_i = y_j]$ is the event that regions i and j depict the same item.

From our training data, we can supply training data $\mathcal{D}_{\texttt{train}}$ in the form of region pairs hand-labelled with their ground truth score $\in \{0, 1\}$:

$$\mathcal{D}_{\texttt{train}} = \{(\mathbf{x}_{ij}, y_{ij}|\xi_i \wedge \xi_j) \quad | \quad 1 \leq i < j \leq M\}. \tag{6.5}$$

At test time we then desire for new unlabelled pairs of regions $(\mathcal{R}_i, \mathcal{R}_j)$, $Pr(y_{ij})$ to be

inferred.

This two-class classification problem is one of the oldest and best-explored examples of machine learning. There are many algorithms which could be used, such as support vector machines, logistic regression or nearest neighbour classification. We use a Random Forest, as in previous chapters. An overview of this training and prediction process is shown in Figure 6.8.

This method follows Vicente et al. [170], who train a Random Forest regressor on pairs of well-segmented and badly-segmented images. The regressor captures features such as the similarity between the foreground regions, and the difference between the foreground and the background. At test time, segmentations of each image in a pair are proposed by a separate algorithm and are jointly scored by the regressor. However, they are limited to only one class of object, and exactly one object instance is assumed per image.

Often the modal class vote is taken as a hard prediction of class membership. Instead we follow [12] in using the fraction of votes for each class to approximate a probability of class membership. p_{ij} can be seen as the parameter of a Bernoulli distribution; i.e. there is a probability p_{ij} that $\mathcal{R}_i$ and $\mathcal{R}_j$ belong to the same class.

At the training stage we take care to deal with the class imbalance in the training data. Hammer et al. [66] showed that when it comes to equivalence constraints, there are approximately K times more negative constraints than positive ones, i.e. $\sum_{ij}[y_i \neq y_j] \approx K \sum_{ij}[y_i = y_j]$. The large class imbalance when K is high can have detrimental effects on the performance of the Random Forest classifier, and can cause a skew in the results. To address this issue we choose an equal sized random subsample of positive and negative pairs to be passed to the classifier for training.

Computing unary scores $Pr(\xi_i \wedge \xi_j)$

To compute the probability that both regions are valid objects, we exploit their independence, i.e.

$$Pr(\xi_i \wedge \xi_j) = Pr(\xi_i)Pr(\xi_j). \tag{6.6}$$

Each unary probability $Pr(\xi_i)$ is computed using the Neural Network formulation of Silberman et al. [153], trained on their NYU dataset. Each region in their dataset is hand-labeled as being either 'furniture', 'wall', 'ceiling' or moveable 'props' such as pillows or bottles, and they train a Neural Network to give a class membership probability for any new segment. The features that they use are colour histograms; mean and standard deviation of colour channels; SIFT histograms; 2D and 3D bounding box dimensions; a pyramid of surface normal histograms; mean, median and maximum planar errors; distance to closest 'wall'; relative depth over the region; and the minimum and maximum heights above the 'ground'. We find that using the probability for being a member of the 'prop' class or the 'furniture' helps to capture the property of being an object. We use the sum of these two class outputs as our estimation for $Pr(\xi_i)$. In effect this is giving the probability that the segment is *not* a segment of wall or ground. An analysis of the performance of this classifier is given in section 6.5.2.

6.3.3 Correlation clustering

It is tempting at this stage to simply place pairs of segments which have a high p_{ij} value in the same cluster. However, our probabilistic interpretation of p_{ij} implies that there is a probability $1 - p_{ij}$ that regions $(\mathcal{R}_i, \mathcal{R}_j)$ should *not* be placed in the same cluster, and it only takes a few false positive connections to cause every $\mathcal{R}$ to be assigned the same label. We therefore need a way of using all our inferred values of p_{ij} to find a clustering robust to false positives. For this task, we use *correlation clustering*.

Given a symmetric affinity matrix $\mathbf{W} \in \mathbb{R}^{M \times M}$, where each entry W_{ij} represents a degree of attraction (positive values) or repulsion (negative values) between data points i and j, a correlation clustering clustering algorithm finds a labelling (or k-way partitioning) of the data points $\mathcal{Y} = \{y_1, \ldots, y_M\}$ which minimizes the energy function

$$E(\mathcal{Y}) = -\sum_{i=1}^{M} \sum_{j=1}^{M} W_{ij} \left[y_i = y_j \right]. \tag{6.7}$$

We follow [3, 4] in interpreting W_{ij} as the log-odds ratio between the probability distri-

bution over positive pairs and negative pairs, i.e.

$$W_{ij} = \log\left(\frac{p_{ij}}{1-p_{ij}}\right) \in \mathbb{R}. \tag{6.8}$$

It follows that if $p_{ij} < 0.5$ then it will favour placing i and j in separate classes, while if $p_{ij} > 0.5$ it will favour placing them in the same class. In contrast to most graph-based pairwise clustering solves a continuous probabilistic degree of similarity is accounted for.

Solving Equation 6.7 for $\mathcal{Y}$ is non-trivial; Bansal et al. [4] prove that finding the optimal clustering when edges are binary is NP-hard. Bansal et al. [4] initially focussed on solving problems with integer edge weights $\in \{-1, 1\}$ using the principle of minimising the number of disagreeing triangles of edges. Bagon and Galun [3] introduced two novel solvers for continuous edge weights based on an extension of the graph cuts algorithm [137], and a third iterative solving method which greedily swaps points between clusters until the energy cannot be lowered any further. It is this last solver, AL-ICM (adaptive-label iterative conditional modes), which we use.

Based on earlier work by Besag [5], AL-ICM is a greedy iterative solver which changes the label of each y_i in turn to the label which it is most attracted to, i.e. which minimises the energy the most. If the data point is repelled by all clusters, then it is assigned a unique singleton label. This is outlined in algorithm 1.

Algorithm 1 AL-ICM correlation clustering solver.

Input: $\mathbf{W} \in \mathbb{R}^{N\times N}$, $\mathcal{Y}_{\texttt{guess}} \in \mathbb{Z}^{1\times N}$, $\texttt{max\_iter} \in \mathbb{Z}^{+}$
Output: $\mathcal{Y} \in \mathbb{Z}^{1\times N}$

```
Y ← Y_guess
iteration_count ← 0
Y_last ← []
while Y ≠ Y_last and iteration_count < max_iter do
  Y_last ← Y
  for each y_i in Y do {Loop over each data point}
    k ← []
    for each y_j in Y do {Summing y_i's attraction to each existing cluster}
      k_{y_j} ← k_{y_j} + W_{i,j}
    if min(k) < 0 then
      y_i ← max(k) + 1 {Creating new singleton cluster}
    else
      y_i ← arg min_j k_j {Assigning to most attracted cluster}
  iteration_count++
```

Clearly each iteration of this algorithm is of order $O(N^2)$; for each of the N points, each other point is summed over to compute the point-to-cluster score vector $\mathbf{k}$. The algorithm does not guarantee convergence and as a greedy solver it is prone to local minima. The initial guess of the labels can therefore have an effect on the final solution. In the original implementation[1] the initial guess of the labels $\mathcal{Y}_{\texttt{guess}}$ is set such that all the items are given the same label. To avoid problems with local minima in this greedy solver we run the algorithm with multiple different values for $\mathcal{Y}_{\texttt{guess}}$ and choose the clustering which gives the lowest energy (Equation 6.7). For our values of $\mathcal{Y}_{\texttt{guess}}$ we choose a vector of all ones (i.e. a single cluster), a vector of unique numbers (each object in a different cluster) and four more vectors of randomly assigned integers $\in [1, N]$.

It is interesting to compare AL-ICM with the k-means algorithm. Like k-means, AL-ICM iteratively assigns each point to the cluster which minimises the energy function the most. However, AL-ICM can also spawn new clusters or erase an existing cluster label on each iteration, which is in contrast to k-means which has a fixed number of clusters to choose from. The 'distance' from a point to cluster in k-means is a distance in L_2 space, while in AL-ICM it is the sum of all the edges from the point to each point in the cluster. Our use of multiple starting guesses for AL-ICM is inspired by the analogous multiple replicate version of k-means.

6.4 Using discovered objects to complete scenes

The output of our object discovery routine is a set of clusters of depth images, each of which we hypothesise represents the same object. We then wish to make use of these clusters to complete the missing regions of our scenes.

One approach to this task could be to register together the 3D points in each cluster to a common coordinate frame, thus forming an object completion. These completions could then be re-inserted into each depth image to complete missing geometry. This task is introduced as *multi-view surface matching* by Huber and Hebert [77]. However, given the limited view of each object in our test data, and the large amount of occlusion and noise, this method could be very brittle and unreliable. (It is worth observing that works in

[1] Available from `http://www.wisdom.weizmann.ac.il/~bagon/matlab.html`

this field, e.g. [77, 84], tend to form completions of objects with highly detailed geometry with many salient regions. This is in contrast to our objects which tend to have large flat surfaces and various forms of symmetry.)

Instead of multi-view surface matching, we use our found clusters to complete our scenes in a different way. We assume that *some* of the scenes observed at test time have had their geometry fully observed. Other test-time scenes have only had a single depth image captured. We can then use our algorithm to find correspondences between objects in the completed scenes and objects in the incomplete scenes. These found correspondences can then be exploited to enable the geometry from the fully completed objects to be used to in-fill the missing geometry in the scenes captured in a single depth image. See Figure 6.2 for an example of this process.

6.4.1 Our multi-view registration algorithm

More formally, we define our depth images captured at test time as belonging to one of two disjoint sets, $\{\mathcal{D}_{\texttt{full}}, \mathcal{D}_{\texttt{partial}}\}$.

- $\mathcal{D}_{\texttt{full}}$ contains depth images captured from scenes which have had their full geometry observed.

- $\mathcal{D}_{\texttt{partial}}$ contains depth images captured from scenes which have only had partial geometry observed; in our case, each of these scenes have been viewed from just a single depth image.

Using our previously described object discovery algorithm, we use the fully-observed objects present in $\mathcal{D}_{\texttt{full}}$ to complete the objects in $\mathcal{D}_{\texttt{partial}}$.

Our object discovery algorithm gives us correspondences between all the *regions* in depth images $\mathcal{D}_{\texttt{full}} \bigcup \mathcal{D}_{\texttt{partial}}$. Given a single depth image $\mathbf{D} \in \mathcal{D}_{\texttt{partial}}$, we use the following algorithm to complete the missing geometry:

- For each region $\mathcal{R}$ in $\mathbf{D}$:

  - We find the set of regions $\{\mathcal{R}_{\texttt{full}}\} \in \mathcal{D}_{\texttt{full}}$ which we hypothesise correspond to the same object as that depicted in $\mathcal{R}$. This step is completed using the algorithms outlined previously in this chapter.

- For each candidate region match $\mathcal{R}_{\texttt{full}}$, we find a 6 degree-of-freedom candidate transformation $\mathbf{T}$ which best brings it into 3D alignment with $\mathcal{R}$. We estimate $\mathbf{T}$ using the iterative closest points (ICP) algorithm [6], initialised to a range of initial starting poses. In our experiments we use 12 starting rotations, each of which is a rotation about the z axis in the range $[0, 30, 60, \cdots, 330]^\circ$. For each starting rotation point we initialise the ICP by aligning the means of the two point clouds. Of the 12 converged transformations, we select as the final candidate transformation the one with the lowest point-to-plane error [22].
- We then have a set of candidate regions, each with an associated transformation; i.e. $\{(\mathcal{R}_{\texttt{full}}, \mathbf{T})\}$. Any of these could be used to complete our scene. We select just one region and transformation pair from this set to use for completion. We choose the pair with the lowest point-to-plane error metric [22], as used in the minimisation of the ICP algorithm.
- Finally, we use the chosen transformation to bring the fully-observed geometry corresponding to region $\mathcal{R}_{\texttt{full}}$ into alignment with $\mathcal{R}$. This gives an estimation of the missing geometry corresponding to $\mathcal{R}$, which is inserted into the final scene prediction.

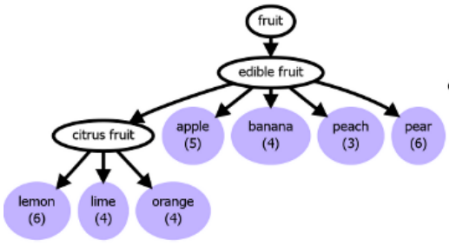
The aggregation of the geometry predictions for each segment gives the final prediction of scene geometry for this depth image.

6.5 Evaluation

First we perform experiments on a turntable dataset, where we evaluate our pairwise matching and clustering algorithms. Next, we evaluate our full object discovery algorithm on a challenging dataset of objects in indoor scenes. Finally, we perform our full object completion pipeline on our tabletop dataset, where we use ground truth data to evaluate completion accuracy.

6.5.1 Turntable dataset for pairwise evaluation

First, we train our model and test on the RGB-D object dataset [101]. The full version consists of over 250,000 views of 300 object types, captured under controlled conditions



(a) Subset of the hierarchy

(b) Example object views

Figure 6.9: The RGB-D dataset. Image (a) shows a section of the hierarchy, while image (b) shows some examples of the objects captured.

	Unique objects	**Total images**	**Images per object**
Training set $\mathcal{D}_{\texttt{train}}$	155	3100	20
Test set $\mathcal{D}_{\texttt{test}}$	50	250	$\mu = 5, \sigma = 1.7$

Table 6.3: The train/test split of the turntable dataset

on a turntable. Each view consists of an RGBD image together with a segmentation mask—see [101] for details of the segmentation algorithm. Because each view comes with a reasonably good object mask, we do not make use of the unary classifier, simply setting $Pr(\xi) = 1$ for testing on this dataset. This means that these first experiments evaluate the *pairwise* scores only.

Each view has both an instance and a category label. Category labels are human-assigned semantic labels, such as `orange`, `hairbrush` or `cereal box`. Within each category, several instances exist; instance labels are absolute, and two images with the same instance label are views of exactly the same object. We use the instance labels as our y_i values.

Train/test split We use a subset of the RGBD dataset as training data $\mathcal{D}_{\texttt{train}}$ and a different subset as test data $\mathcal{D}_{\texttt{test}}$. Our train/test split is shown in Table 6.3. We want to demonstrate the generalizability of our system in grouping object types which did not form part of the training set. To show this effectively, we ensure that our test set does not contain any overlap with the training set at either the instance or category level. For example, a 'notebook' instance is included in the training set; therefore no notebooks are present in the test set.

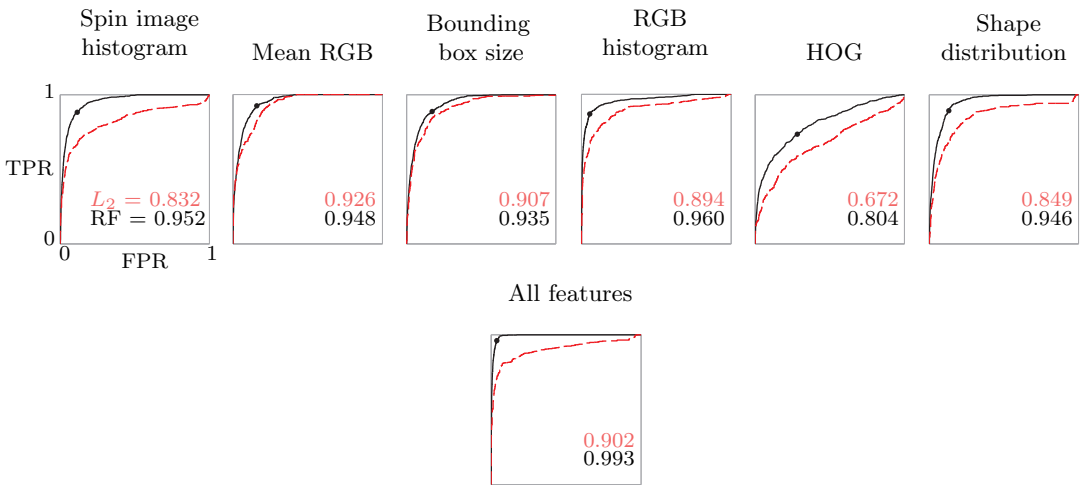


Figure 6.10: Receiver operating characteristic (ROC) curve comparing feature types for pairwise classification accuracy on the RGBD dataset. The black dots on the curves represent a threshold of 0.5. Black lines represent Random Forest results; red dotted lines the results from logistic regression on L_2 distances.

For each pair of test objects, we have a ground truth pairwise class label $y_{ij}^{\mathtt{GT}} \in \{0, 1\}$ and a predicted output from our Random Forest classifier $Pr(y_{ij}|\xi_i \wedge \xi_j)$. We show the accuracy in Figure 6.10. Numbers inside each plot refer to the area under the curve (AUC); higher is better, and 1 is perfect.

The histogram of gradient feature has a particularly poor performance, while using the mean RGB has the highest score. This is unsurprising given the highly controlled conditions under which the objects were captured.

To show the benefit of our use of Random Forests above a method operating in Euclidean space, we also train a logistic regressor on the L_2 distances $\mathbf{x}_{ij}$. These are also plotted in Figure 6.10 as dotted red lines. The difference between this L_2 distance measure and the Random Forest result is considerable, showing the benefits brought from the non-linear, randomised forest.

We present some of the specific pairwise results in Figure 6.11. This reveals some of the difficulties present in performing comparisons between single RGBD views. Figure 6.11(c) shows the same object (a camera) viewed from wildly differing camera angles. Furthermore, both views are missing a large amount of depth data. These data discrepancies have contributed to a false negative result.

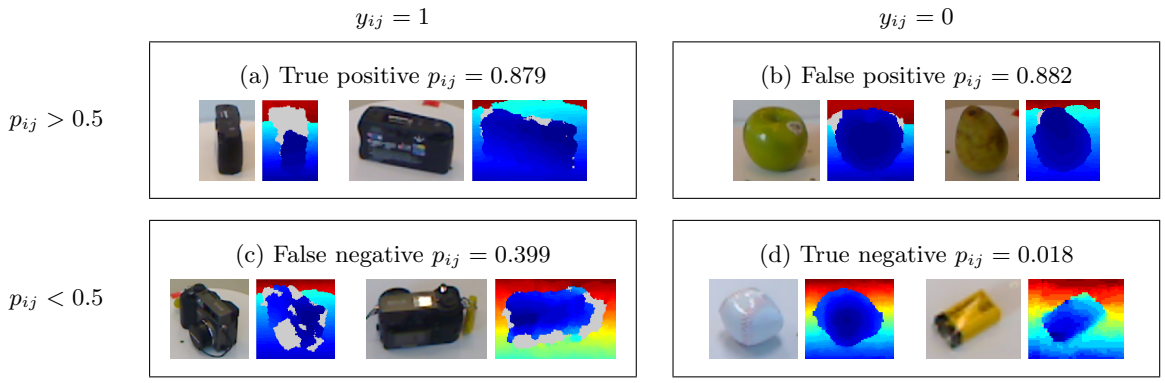


Figure 6.11: Examples of true positive, true negative, false positive and false negative matches using the random forest classifier

Qualitative clustering results

Figure 6.12 shows exemplar results from our clustering algorithm. The same clustering results are shown in a confusion matrix in Figure 6.13.

As well as the qualitative clusters, we also report a per-cluster purity measure. The purity of a cluster is the fraction of members which belong to the cluster's modal ground-truth class. 28 out of the 40 found clusters have a purity of 1. Most of the clusters with purity < 1 make errors at the category-instance level, for example by grouping together objects which come from the same category but which are not the same instance. In many of these clusters, the distinction between the instance groupings is very difficult even for a human eye. The green apples present in cluster (k) from Figure 6.12, for example, consist of two different instances, as do the bananas in Figure 6.12(ah).

Some of these clusters contain arguably graver errors, where objects which differ at the category level are erroneously grouped. This is visible in Figure 6.12(m), where cameras are confused with mugs, (q), where cameras are muddled with mobile phones and (ab), where apples are mixed in with peppers. In cluster (k) a lone orange pepper is grouped with a collection of apples. This is an indication of problems in pairwise matching; more will come when we attempt to perform similar clustering on sections of RGBD scene data, where most of the segments should in fact be placed in singleton clusters.

As well as the problems with some of the clusterings, there are interesting successes. Clusters (h) and (n) appear to the human eye to contain views of the same football. However, inspection of the confusion matrix reveals that they are in fact different ball

types, which the algorithm has correctly managed to identify.

Cluster purity does not penalise clusters which are erroneously split. A trivial way to get high purity values would be to place each object in its own distinct class. We therefore make use of the adjusted Rand index to give an overall score in future evaluations, as this takes into account errors in the merging of classes and errors in the splitting of classes. The adjusted Rand index (ARI) [78] uses the ground truth labelling and the inferred labelling to compute a single score evaluating the quality of the clustering, where an ARI of 1 indicates a perfect partitioning, while an ARI of 0 represents a partitioning no better than chance. We give details of the computation of the ARI in Appendix D.

Comparison with other clustering algorithms

We compare our correlation clustering algorithm with alternative methods which do not rely on training data. The aim is to show the benefits which the supervision brings to the quality of the clustering, and to show that supervision can allow us to form good clusters without the need for a tunable parameter. Each comparison algorithms accepts a single parameter λ. We use:

k-means [108] due to its age and ubiquity in machine learning applications. λ_{kmeans} is the number of clusters to be found. We test each possible value; i.e. $\lambda_{kmeans} = [1, 2, \ldots, M]$.

Spectral clustering [118] due to its recent use in object discovery algorithms. $\lambda_{spectral}$ is the number of clusters to be found, and we select the same range of $\lambda_{spectral}$ as for λ_{kmeans}. The variant of spectral clustering we use is *multiclass spectral clustering*, and we use Euclidean distances between data points to form the distance matrix.

Agglomerate clustering [54], also due to its recent use in object discovery algorithms. The parameter $\lambda_{agglomerate}$ is a measure of cluster self-similarity—for this we linearly interpolate 50 values between 0.01 and 5.

Algorithms which operate in Euclidean feature space, such as k-means, are strongly affected by the relative scaling of features. To provide a fair comparison, we perform some experiments using only single features. When using all features we scale each feature (e.g.

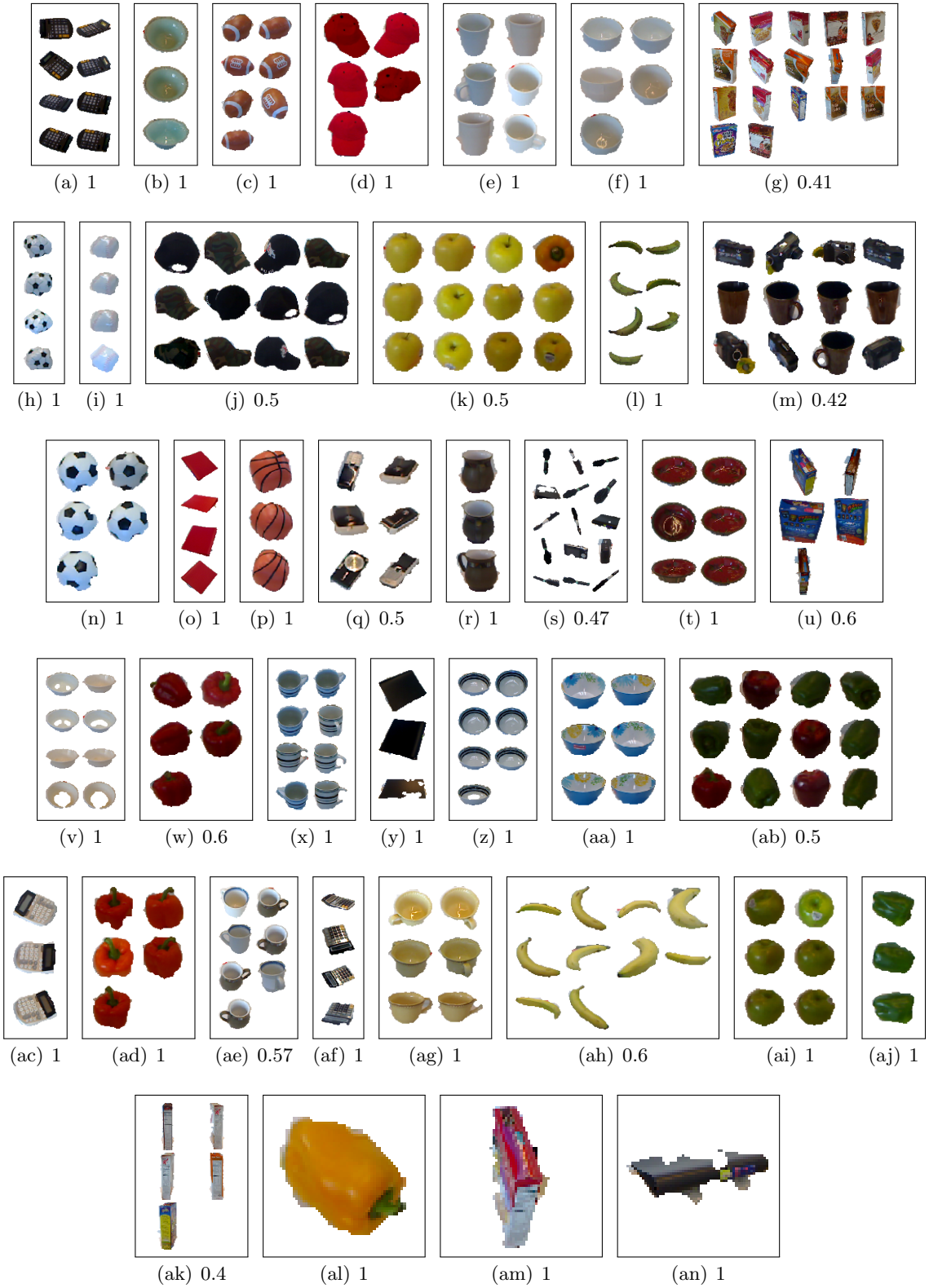


Figure 6.12: Qualitative clustering results. Objects in each rectangle are one cluster found by our algorithm. Numbers below each rectangle represent cluster purity. The purity of a cluster is the fraction of members which belong to the cluster's modal ground-truth class. This clustering scored an adjusted Rand index of 0.664. This is a less than perfect clustering result, which comes about due to errors which can more clearly be seen in the confusion matrix in Figure 6.13.

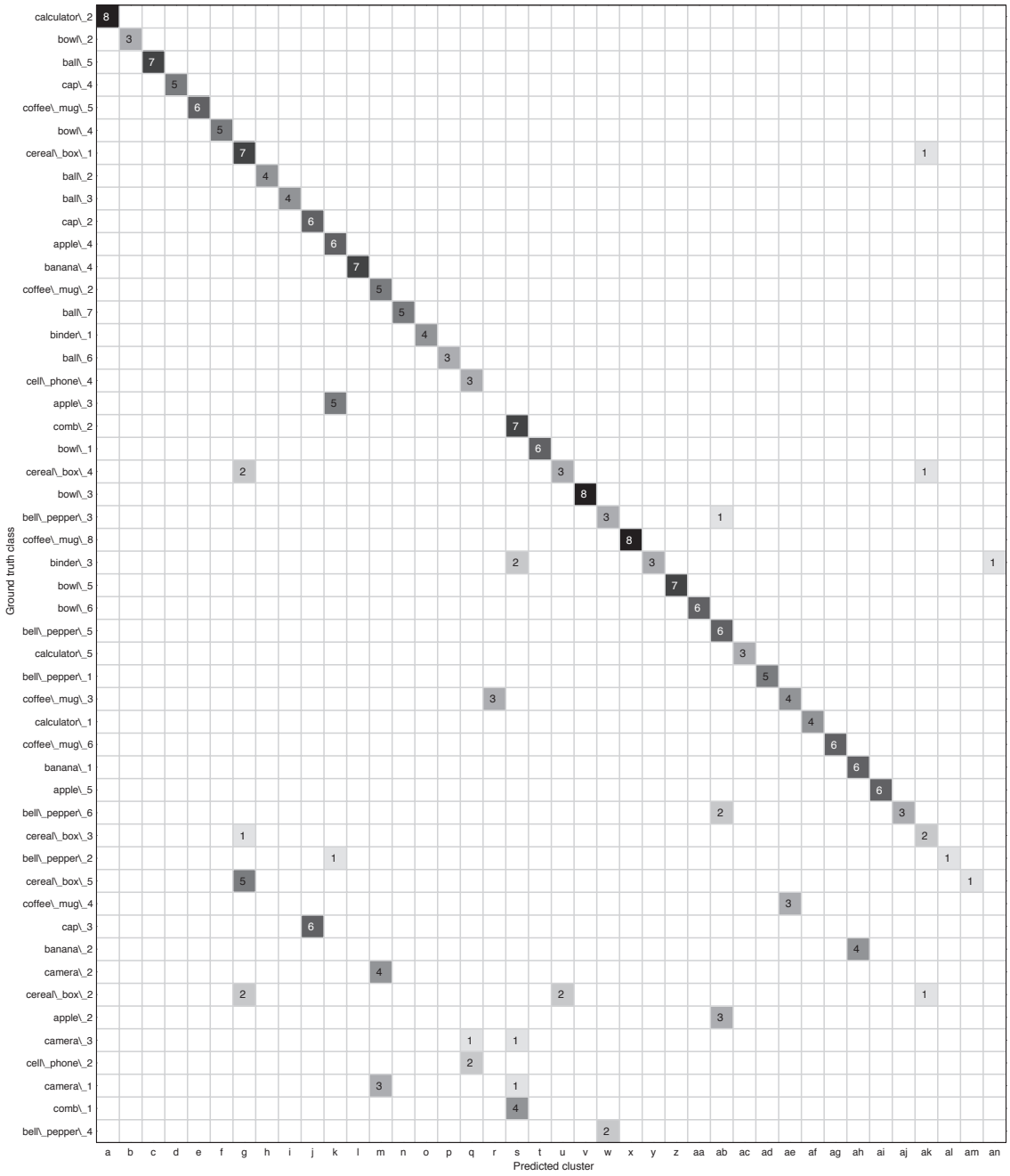
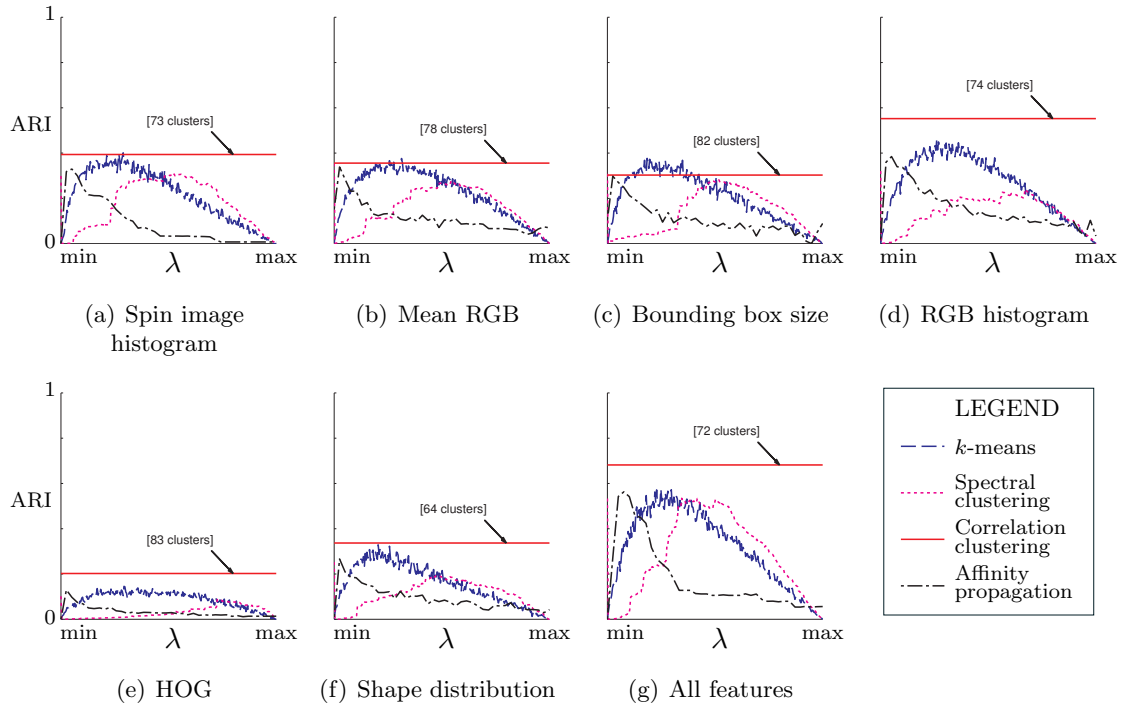


Figure 6.13: Confusion matrix for the object discovery performed on dataset $\mathfrak{D}_1$, using all features. Each row represents a ground truth class, each column a found cluster; numbers in the grid represent assignment of items to ground truth and found clusters. A perfect result would be a square matrix with only on-diagonal entries. Clearly our found clustering has fewer labels than the ground truth clustering. Labels for the columns correspond to subplot labels in Figure 6.12.



(a) Spin image histogram (b) Mean RGB (c) Bounding box size (d) RGB histogram

(e) HOG (f) Shape distribution (g) All features

Figure 6.14: Comparison of our correlation clustering implementation with other clustering methods, over a range of their parameter settings (see Section 6.5.1 for details of λ settings). Results are shown for dataset $\mathfrak{D}_1$; each plot (a)-(f) corresponds to a different feature. The number of clusters recovered by our method (red line) is indicated. The ground truth labelling has 10 clusters. Higher values for adjusted Rand index are better.

'spin image histogram') to sum to one. We found this gave superior performance in the comparison algorithms than normalizing each dimension of $\mathbf{x}_{ij}$ to $\mathcal{N}(0,1)$.

Results using single features to cluster segmented objects in $\mathcal{D}_{\texttt{test}}$ are shown in Figure 6.14. For this dataset, only the spin image histogram, mean RGB and bounding box size ever outperform our method, and even then only with a very limited range of λ values. The RGB histogram appears to be the feature with the most discriminatory power. This is most likely due to the highly controlled conditions under which this data was captured [101]. We then combine all features and repeat the experiment. The results for this combined set is shown in Figure 6.14. Our algorithm maintains a higher performance over our competitors. As with the pairwise ROC results, the combination of all features boosts performance above any individual feature in isolation.

6.5.2 Real-world dataset

We now evaluate our object discovery algorithm on a challenging real-world dataset. $\mathcal{D}_2$. This dataset comprises a subset of 21 frames from the 8 video sequences accompanying the

(a) (b) (c) (d)

Figure 6.15: Some qualitative segmentation results. Each different coloured region in the right-hand image of each pair represents a different region $\mathcal{R}$, while white areas represent missing data or regions identified as dominant planes.

RGBD object dataset [101]. We augment these images with 15 new RGBD images, representing more views of household objects in indoor scenarios (see Figure 6.15 for examples). The frames are mostly non-overlapping, although some view the same environment but from highly differing angles.

Segmentation results

We present some qualitative segmentation results in Figure 6.15. While some objects can be seen to be well segmented (such as the cap in Figure 6.15(a) and most of the objects on the table in Figure 6.15(b)), there are several failure cases. We can categorise these failures as so:

Over-splitting One cause of failure is an object being erroneously split into two regions; this occurs in the case of the turquoise bowl in Figure 6.15(b) and the laptop in Figure 6.15(a).

Merged objects Another failure case is two objects being merged into one region. This occurs with the tea box and the microwave behind it in Figure 6.15(c), and the bowl and the sink in Figure 6.15(d).

Lost parts Some of the regions are incomplete representations of objects, either due to

missing data or part of the region being incorrectly identified as being 'background'.

False positives for object proposals The final, and arguably most detrimental cause of failure is that the majority of the proposed regions do not represent an object at all, but instead are some 'background clutter', or part of a far larger piece of furniture or structure. This is particularly apparent on the border of images (a-c). We aim to partially deal with this through the use of our unary scoring.

Quantitative evaluation

We map our ground truth region labelling to the regions detected from our segmentation algorithm. We assign a label $\hat{y}_i$ to region $\mathcal{R}_i$ according to the rule

$$\hat{y}_i = y_k \iff \frac{|\mathcal{R}_i \cap \hat{\mathcal{R}}_k|}{|\mathcal{R}_i \cup \hat{\mathcal{R}}_k|} > 0.7, \tag{6.9}$$

where $\hat{\mathcal{R}}_k$ is a ground truth object boundary with label y_k. Regions which are left unassigned to any class are given a special label -1.

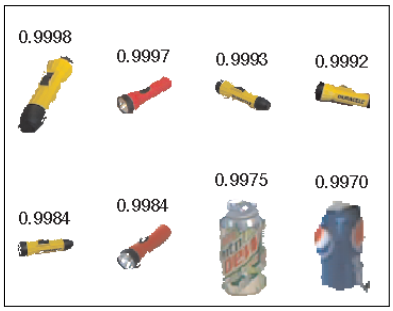
By analysing these labels we can make a quantitative evaluation of our segmentation algorithm. Of the 105 instances of objects, 85 were segmented well enough to be given a label by (6.9). Of these 85 regions, 9 had unique object labels not shared with any other region. In addition to these 85 regions, there were 94 regions found which did not share a good enough overlap with a ground truth object to be given a label, according to (6.9).

In our final dataset 52.5% of the segments can thus be considered to be 'noisy'.

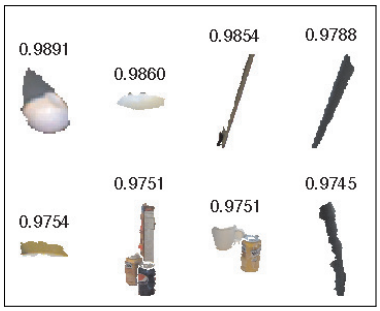
Accuracy of unary score prediction

We assessed the quality of $Pr(\xi)$ classifier (Section 6.3.2) at discriminating between 'object' and 'clutter'. We define a region as 'clutter' if it is not given a label under equation 6.9. We can use our prediction $Pr(\xi)$ and the ground truth clutter labelling (where $\hat{y}_i = -1$) to plot a ROC curve, which we show in Figure 6.17. The area under the curve is 0.754.

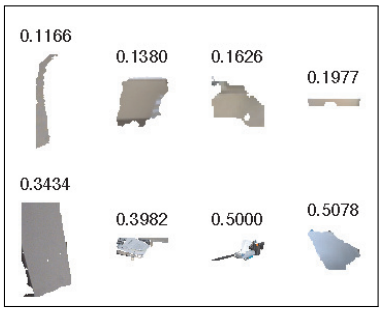
We show examples of true positives, true negatives, false positives and false negatives in Figure 6.16. From this sample of results we can see that the unary score does some good in helping to separate the classes; the top scoring segments are clearly all reasonably well-segmented objects, while the lowest scoring segments are on the whole parts of wall,



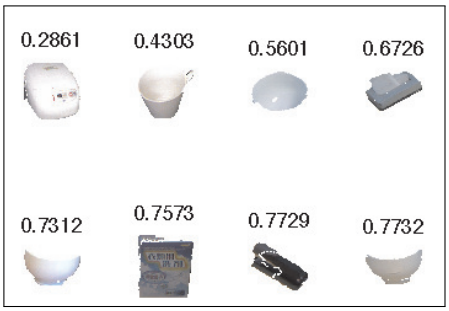
(a) Highest scoring objects ('True positive')



(b) Highest scoring non-objects ('False positive')



(c) Lowest scoring non-objects ('True negative')



(d) Lowest scoring objects ('False negative')

Figure 6.16: Examples of true positive, true negative, false positive and false negative unary scores. Numbers with each image represent the assigned unary score $Pr(\xi_i)$.

floor or multiple objects joined into one segment. Some low-scoring segments are in fact well-segmented objects; these false negatives are show in Figure 6.16(d).

Evaluation of pairwise results p_{ij} on dataset $\mathcal{D}_2$

False negatives in the pairwise matching $Pr(y_{ij})$ can occur when two regions depict the same object but with very different segmentations, or viewed under different lighting conditions (e.g. Figure 6.18(d)). False positives most frequently occur when the unary probabilities $Pr(\xi_i)$ are incorrectly high, causing pairs of background clutter to be incorrectly given a high p_{ij} value—see Figure 6.18(c) for an example of this.

We can assess the quality of our pairwise results in comparison to the ground truth pairwise scores. Firstly we analyse the quality of the individual pairwise scores $Pr(y_{ij}|\xi_i \wedge \xi_j)$, taken from the Random Forest classifier; we find the area under the ROC curve to be 0.9733. Next, we analyse the pairwise scores when combined with the unary scores, i.e. $Pr(y_{ij} \wedge \xi_i \wedge \xi_j)$. Including the unary score increases the area under the curve to 0.9756.

A receiver-operator characteristic curve for each of these measures is plotted in Figure

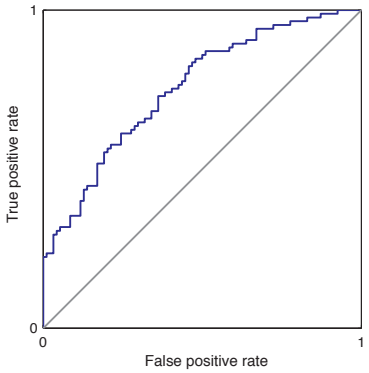


Figure 6.17: Receiver-operator characteristic curve for unary scores $Pr(\xi_i)$. The area under the curve is 0.754

	$y_{ij} = 1$	$y_{ij} = 0$
$p_{ij} > 0.5$	(a) True positive (75) 	(b) False positive (638) 
$p_{ij} < 0.5$	(c) False negative (13) 	(d) True negative (15,205) 

Figure 6.18: Pairwise results examples, showing some examples of class of pairs based on their scores p_{ij}; the values for p_{ij} include both the pairwise and the unary scores. Numbers in brackets indicate the number of pairs falling into each category.

6.19. The most interesting difference between the two curves plotted is the position of the dots representing the threshold value, at $p_{ij} = 0.5$. Including the unary scores naturally lowers the values of each p_{ij}, and causes the false positive rate to decrease at the expense of a lower true positive rate; in many ways one big positive effect of the unary scores is to shift the inherent threshold used in the log-odds ratio conversion (equation 6.8). As we will see, this has a large positive effect on the quality of the recovered clusters.

Qualitative clustering results

We then evaluate our clustering algorithm on the segments. The recovered clusters with two or more members are presented in Figure 6.21. Some results in the context of the original RGB-D images are delineated in Figure 6.20. We present a confusion matrix comparing the clusters we recovered with the ground truth clustering in Figures 6.22.

Failures can occur for a number of reasons. Enough false positives cause errors in the final clustering. The cluster shown in Figure 6.21(a), for example, is formed of background

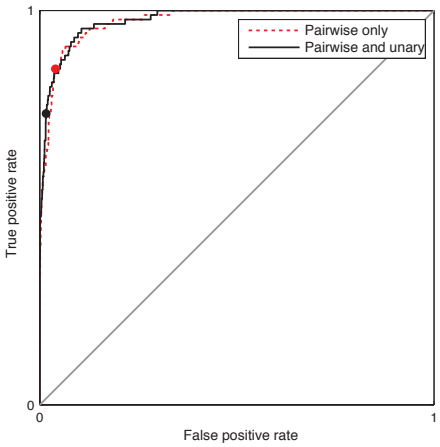


Figure 6.19: Receiver-operator characteristic curve for pairwise scores; the red dashed curve shows results $Pr(y_{ij}|\xi_i \wedge \xi_j)$ from the Random Forest classifier, and the black curve the same results after includion of the unary scores, i.e. $p_{ij} = Pr(y_{ij} \wedge \xi_i \wedge \xi_j)$. The dots show the position of the threshold at 0.5.

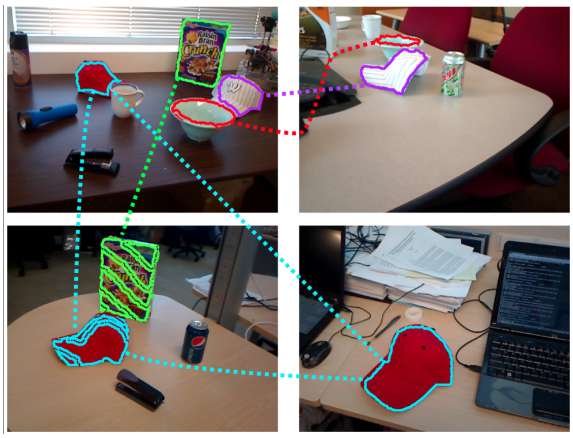


Figure 6.20: Some objects found shown in the context of the original images.

segments which score highly on both the conditional pairwise term $Pr(y_{ij} \mid \xi_i \wedge \xi_j)$ and the unary terms $Pr(\xi_i)$ and $Pr(\xi_j)$. In this instance, forming a better estimate of $Pr(\xi_i)$, perhaps trained on scenes more similar to our test data, could help prevent these kinds of errors.

Quantitative clustering evaluation

As we now have a ground truth label for each segment, we can quantitatively evaluate our clustering algorithm. We perform the same evaluation as we did for segmented objects, comparing the clustering algorithm over a wide range of competitors' parameter values. Results are presented in Figure 6.23.

As with our work on segmented regions we outperform all the other comparison clustering methods at every choice of their parameter values. The overall performance is lower than we saw with the turntable objects. The difference between the training and test dataset has had a large impact on the quality of the pairwise scores, and the inclusion of background clutter means that many non-object segments are erroneously included in clusters. We can see here the positive benefit of unary scores. These help to prevent items of background clutter becoming included in clusters, and this has a large net benefit on the clustering quality.

6.5.3 Timings and implementation details

Our segmentation routine was coded in MATLAB and took on average 1.59s per image, while our feature computation, also MATLAB, took 10.8s per image. The Random Forest classification and correlation clustering took 1.26s and 10.8s respectively when run on dataset $\mathfrak{D}_2$. All experiments were performed on a machine with a 2.4 GHz Intel i5 processor, with 8 GB of RAM.

6.5.4 Evaluation of object completion algorithm

In this section we evaluate the success of our algorithm in recovering the missing geometry of scenes, using the full geometry completion pipeline described and evaluated previously in this chapter. To compare to our alternative approaches presented in previous chapters we

perform object completion on our tabletop dataset. We use the same model as described in Section 6.4 to perform the object discovery on the tabletop dataset.

Our completion algorithm assumes that some of the scenes seen at test time have had their full geometry captured. To enable this, we divide our test time scenes into two equal-sized folds, each comprising 15 scenes (and 60 depth images). We then run the object completion algorithm twice. The first time round, the first fold has the full geometry of each scene while the second fold is made up of single depth images to be completed. On the second iteration the roles of the two folds are flipped. In this manner, we are able to complete each scene in our test set from a single depth image as input.

Completions of some scenes are shown in Figure 6.24. We note that many scenes are completed to a high degree of fidelity. Rows (a), (b), (c) and (e) are completed very well, with details and smoothness of objects matching the ground truth. Details such as the kettle handle are recovered well. Some quantisation artefacts from the geometrical transformations are present; these are especially visible on curved objects, e.g. the kettle in (e).

Where failures occur they are more catastrophic than failures in previous chapters. The milk carton in (d) is badly completed due to an incorrect transformation found to align the two objects. Similarly, the head in (f) is misaligned by 90 degrees around the z-axis, though this is less harmful to the actual occupancy evaluation at that point. (h) shows a failure which is present due to a failure in the object discovery algorithm. The geometry of the head is aligned onto both the cereal box and the smaller tea box in the scene, as they were erroneously placed in the same cluster.

Quantitative results In Table 6.4 we evaluate our completions, following the same methodology as in previous chapters. This algorithm has the highest IoU across all the alternative algorithms and baselines. This is hardly surprising, as in this algorithm we give the algorithm access to full, 'ground-truth' completions of each object in the test scene. In fact, it is somewhat surprising that this algorithm did not perform better, given the data.

	IoU	Precision	Recall
Object discovery completions	**0.712**	0.863	**0.796**
Structured completions	0.658	0.811	0.717
Implicit (Offset)	0.522	0.722	0.666
Implicit (AVOF)	0.625	0.819	0.732
Implicit (AVOF + Offset)	0.627	0.823	0.729
[178] (t=2)	0.528	0.773	0.630
[178] (t=3)	0.378	**0.853**	0.405

Table 6.4: Quantitative evaluation of the object discovery completion algorithm, evaluated on our tabletop dataset.

6.6 Conclusions

We demonstrated in the first part of this chapter how training data can be used to cluster together previously unseen object classes, using a recently proposed solver for correlation clustering together with a pairwise Random Forest classifier. This system achieves better clustering results than k-means, spectral clustering and agglomerate clustering, even when optimum values for their parameters are chosen.

It is important to note that while the training and test images depicted separate classes and instances, there were all captured in the same fashion, i.e. using a turntable and a Kinect sensor.

We have furthermore demonstrated our supervised correlation clustering algorithm on cluttered scenes. We used a basic segmentation algorithm to hypothesise regions corresponding to objects before assessing the probability of each pair of regions sitting in the same cluster, using a Random Forest trained on images from the well-segmented RGBD dataset. To deal with the effects of non-object images in the test set we introduced a unary term into the pairwise probabilistic formulation, which measures the probability that each region is an accurate view of an object. We show that this method outperforms traditional clustering techniques, at all settings of their parameters.

Finally, we demonstrate that this object discovery algorithm can be used at test-time to form completions of scenes.

6.6.1 Limitations and opportunities for future work

There are many opportunities for improvement and exploration based on the current algorithm. One large area for improvement is in the unary scoring which we have seen to give some erroneous results under qualitative and quantitative analysis. Our pairwise measures are also limited by our training data, which does not capture objects under widely varying poses. We would expect that including more diverse training data more similar in variety to our testing data would help improve the pairwise measures.

An interesting opportunity for further development could be match regions not of depth images, but of partial or complete meshes of scenes, for example as formed by a Kinect Fusion scan [80].

Pixel-perfect segmentation Kang et al. [88] address the problem of finding good, pixel-perfect segmentations by using an iterative approach, where objects found on the first pass of their algorithm are used to help improve the segmentation on the second iteration via a graph-cuts method. We could use this concept to help us to improve the quality of the segmentations produced by our algorithm.

Computational complexity The number of pairwise matches is of order $O(N^2)$ in the number of items being clustered. This is clearly a major limitation to the ability to scale the algorithm to large collections of images; 1000 images each split into 50 regions would generate 1.25×10^9 pairwise relations to be evaluated. This large number of pairwise matches is expensive to compute, and then increases the complexity of clustering routines operating on these matches. Approaches to deal with this scaling problem could include:

- ⇛ Recent works have presented promising approaches to improve the tractability of correlation clustering on large datasets. For example, Chierichetti et al. [23] demonstrate a scalable route to an approximate solution using the MapReduce paradigm.
- ⇛ To prevent the need for $O(N^2)$ evaluations of the Random Forest, we propose a 'pre-clustering' method. After the segmentation step of the algorithm, each image region is placed into one *or more* bins based on a simple heuristic. Edge-wise computation and correlation clustering is then performed within each bin, before all the results are aggregated.

For example, one bin might contain all segments which are less than 30cm on their longest length, while another might contain all segments which are greater than 20cm on their longest length. This proposed method would ensure that it is not necessary to compute edges between *all* pairs of regions, only between pairs of regions which have any chance at all of ultimately being placed into the same cluster (Figure 6.25). A similar naive pre-clustering is performed by Liu et al. [105] and Kang et al. [88].

Figure 6.21: Clusters found from our real-world dataset. All clusters of size two or more are depicted.

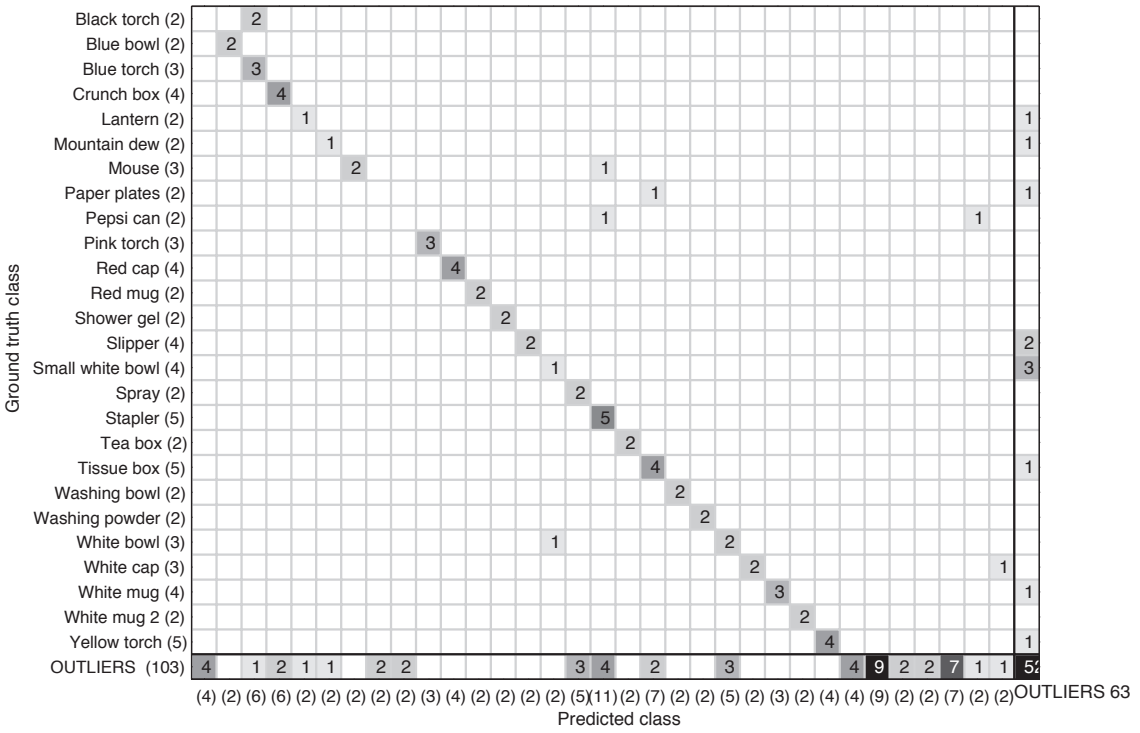


Figure 6.22: Confusion matrix for the object discovery performed on our scene dataset, using all features. Each row represents a ground truth class, each column a found cluster; numbers in the grid represent assignment of items to ground truth and found clusters. A perfect result would be a square matrix with only on-diagonal entries. Outliers (i.e. singleton segments) are combined into the final row (ground truth outliers) and final column (predicted outliers)—otherwise, the ordering of rows and columns is arbitrary. Numbers in brackets on x and y axis represent total size of predicted and ground truth clusters respectively.

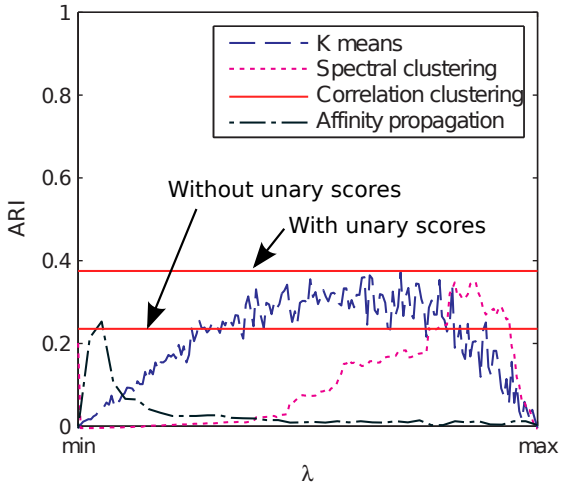


Figure 6.23: Comparison of the correlation clustering algorithm with alternative clustering methods. Results are shown with both the inclusion and exclusion of the unary terms.

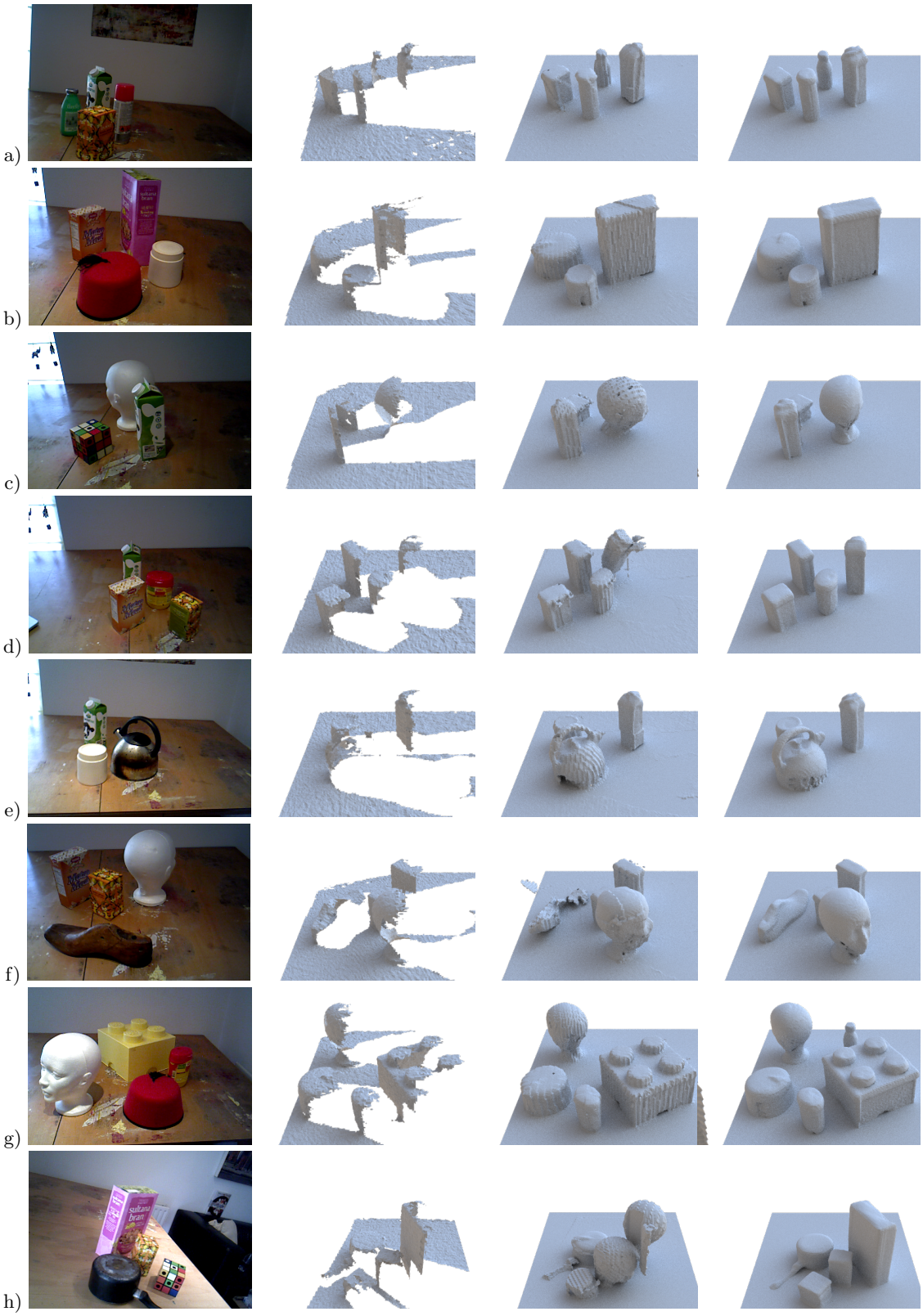
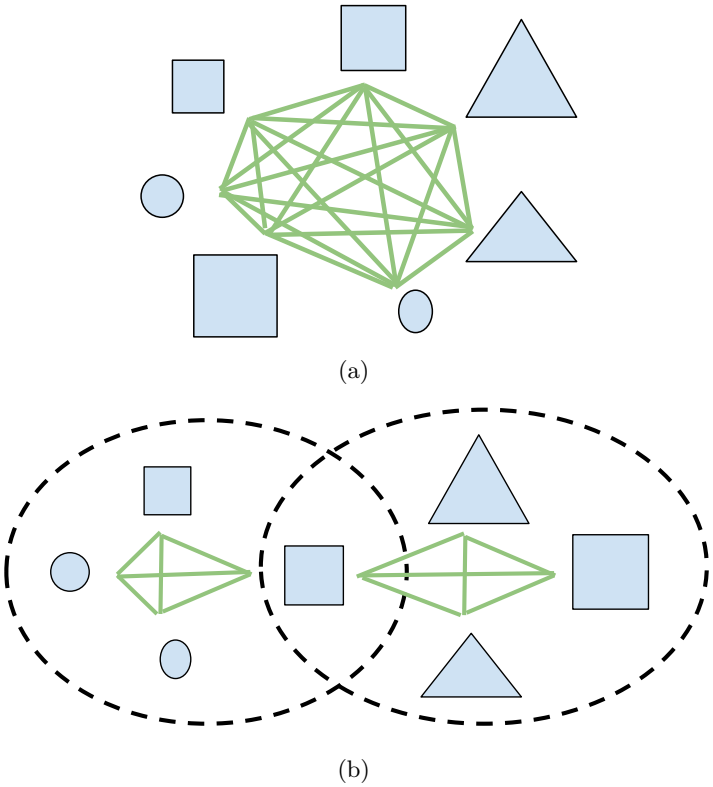


Figure 6.24: Qualitative object discovery results on the tabletop dataset



(a)

(b)

Figure 6.25: An illustration of how 'pre-clustering' could reduce the complexity of the algorithm. In (a) a measure is computed between each pair of objects making a total of 21 pairwise measures. In (b) the objects have been naively sorted into two bins (shown as dashed ovals) according to their size. Note that the medium-sized square appears in more than one of these bins. Now, correlation clustering can be performed within each of these bins. In (b), only 12 pairwise measures have been computed.

Chapter 7

Conclusions

A long-term goal in computer vision has been to gain a 3D model of the world. This has broad applications, from robotic navigation and grasping to image editing and virtual reality. However, to gain a full 3D model we typically require an imaging device to capture images of a scene from multiple viewpoints. In this thesis we considered the scenario of reconstructing the full 3D shape of indoor scenes, but given as input a single depth image. In Chapters 4 and 5 we assume just a single test-time depth image, while Chapter 6 operates under the assumption that some test-time scenes have their full geometry recovered.

Due to the ill-posed nature of the problem, we formed statistical models over the possible occupancy values of each voxel, making use of a set of training data (and carefully designed algorithms) to learn a mapping from our depth image to voxel states.

7.1 Our occupancy prediction methods

Our first contribution is the introduction of two new datasets which can be used for evaluation of voxel occupancy prediction algorithms. We evaluated the difficulties with using existing datasets for such a purpose, as very few have been designed to capture the full geometry of the scene. We believe the datasets will be useful beyond the task of geometry prediction however; in particular, the synthetic dataset has segmentation masks available, and the generation algorithm can be adapted to generate datasets for many different applications.

In Chapter 4 we develop an algorithm which makes a prediction of occupancy for each

voxel in an unknown state in a scene. Using a traditional machine learning pipeline, we develop features which encode the position of each voxel relative to the observed surfaces. We demonstrate that we can use this to form excellent predictions of geometry in the scenes, outperforming state-of-the-art baselines.

We note, though, that this per-voxel algorithm does not make clean and smooth predictions of geometry. While numerical scores are high, qualitative inspection of the predictions reveal that the results tend to be sparse and noisy. From this observation we motivate the *structured prediction* algorithm we present in Chapter 5. Here, we extract as training data cuboid regions of each training scene — these are then used to make predictions at test time. This resulted in much more clean and regular predictions than those made in Chapter 4. We also note that this structured prediction algorithm contains several design choices, such as the size of the voxlets and the number of predictions to make at test time.

Of these two algorithms, the one which numerically performed the best actually varied according to dataset (Table 7.1). This suggests that each method may have its own merits depending on the dataset. Finally, in Chapter 6 we present a very different method for completing scenes based on the discovery of repeating objects in the test set of multiple images. In combination with the depth data, and a learning algorithm, we apply a supervised clustering algorithm to learn matches between regions from the depth images. We use these clusters to complete the geometry of objects from incomplete scenes, using the fully-observed geometry from other scenes containing the same object.

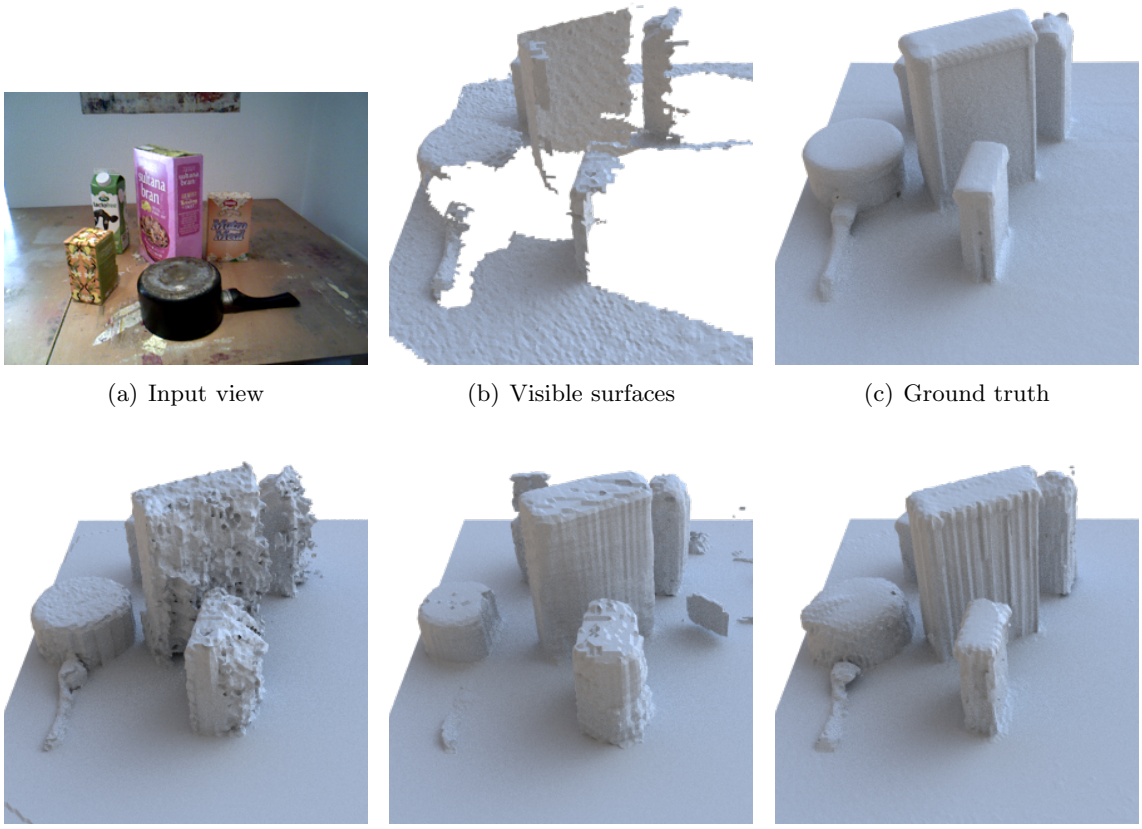
In Table 7.2 we review some of the advantages and disadvantages of the methods presented in this thesis. In Figure 7.1 and Figure 7.2 we see how our methods compare on the same scene. The key points from Table 7.2 are apparent: Our per-voxel predictions are noisy but largely accurate, while voxlets form smoother predictions. Completing objects using object discovery works well where good correspondences have been found, but fails catastrophically otherwise.

	Tabletop dataset			*Synthetic dataset*		
	IoU	**Precision**	**Recall**	**IoU**	**Precision**	**Recall**
Chapter 4						
Implicit (Surface)	0.522	0.722	0.666	0.716	0.870	0.797
Implicit (AVOF)	0.625	0.819	**0.732**	0.769	0.893	**0.838**
Implicit (AVOF + Surface)	0.627	0.823	0.729	**0.771**	0.906	0.831
Chapter 5						
Structured completions	**0.658**	0.811	0.717	0.737	0.864	0.833
Chapter 6						
Object discovery	***0.712***	0.863	***0.796***	-	-	-
Baselines						
Zheng et al. [178] (t=2)	0.528	0.773	0.630	0.645	0.931	0.677
Zheng et al. [178] (t=3)	0.378	**0.853**	0.405	0.322	**0.968**	0.326

Table 7.1: Quantitative evaluation of all algorithms on our datasets. Numbers in **bold** are the best across all algorithms given the standard train/test split. Where the object discovery algorithm, which operates under different test-time data assumptions, outperforms other variants, its result is additionally given in ***bold-italic***.

	Advantages	**Disadvantages**
Chapter 4: Per-voxel, implicit completion	• Object agnostic, and generalises well to new shapes • Efficient to train and test • Can easily generalise to multiple input images • Naturally respects observed geometry	• Predictions are noisy and lack structure • Surface continuation not explicitly encouraged
Chapter 5: Structured voxlet completion	• Smooth, visually plausible predictions • Replicates structure of real world	• Hard to generalise too far beyond shapes in training set • Several parameters to be decided, e.g. size of voxlets • Hard to force to respect the real-world geometry
Chapter 6: Object discovery for completion	• Accurate object completions	• Relies on seeing objects multiple times in image set • Many steps to algorithm, each of which can be a point of failure • Failures can be catastrophic

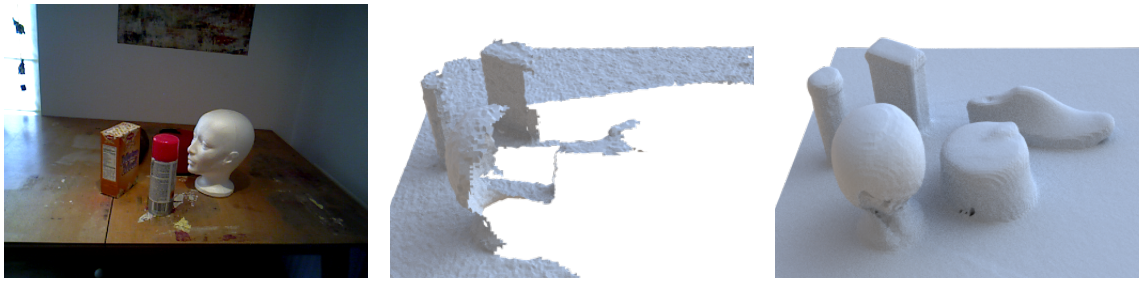
Table 7.2: Some of the key strengths and weaknesses of the methods for occupancy prediction presented in this thesis.



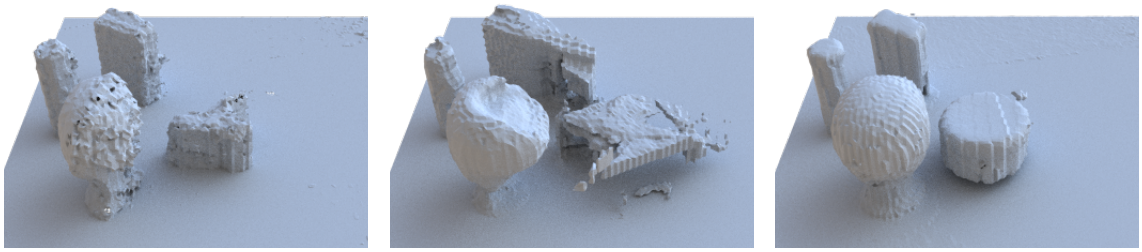
(a) Input view (b) Visible surfaces (c) Ground truth

(d) Chapter 4 — Implicit (IoU = 0.725) (e) Chapter 5 — Voxlets (IoU = 0.751) (f) Chapter 6 — Object discovery (IoU = 0.843)

Figure 7.1: A comparison of algorithms on image 00207_[536] from the tabletop dataset



(a) Input view (b) Visible surfaces (c) Ground truth



(d) Chapter 4 — Implicit (IoU = 0.530) (e) Chapter 5 — Voxlets (IoU = 0.455) (f) Chapter 6 — Object discovery (IoU = 0.658)

Figure 7.2: A comparison of algorithms on image 00218_[121] from the tabletop dataset

7.2 Analysis of evaluation metrics

We have evaluated our methods quantitatively using the intersection over union, precision and recall between our predictions and the ground truth voxel occupancy (Section 2.4). These give a good overview of the quality of the predictions. However, as can be seen from Figure 7.1, results can have similar scores but can look very different.

For different application areas we may consider a more specific evaluation metric, capturing the qualities required for that purpose. For example:

- For path planning we may not care about how 'realistic' the predictions look, but we may have a problem if the prediction suggested we could manoeuvre our vehicle to a position which is in fact inaccessible from our starting point. We may therefore consider an evaluation metric measuring the difference between the path predicted based on the occupancy prediction, and the actual best path given the ground truth occupancy.

- Similarly, for a robot grasping application the important aspect is that the robot arm doesn't collide with any of the objects in the scene when picking up a target

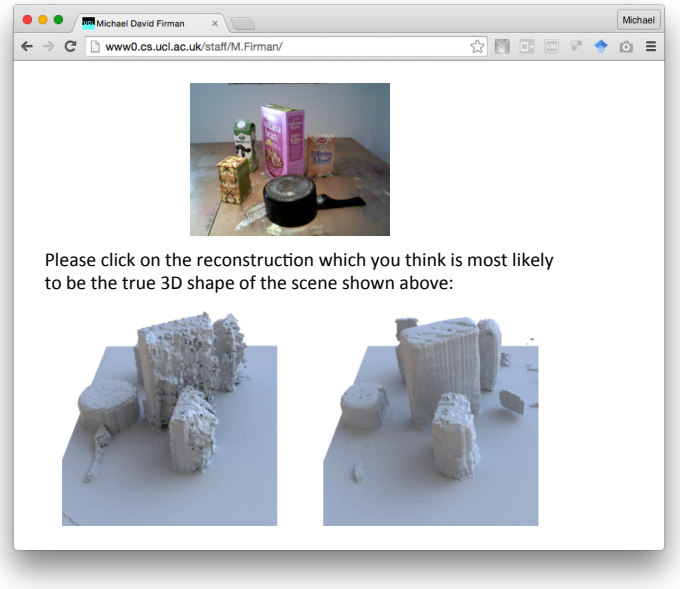
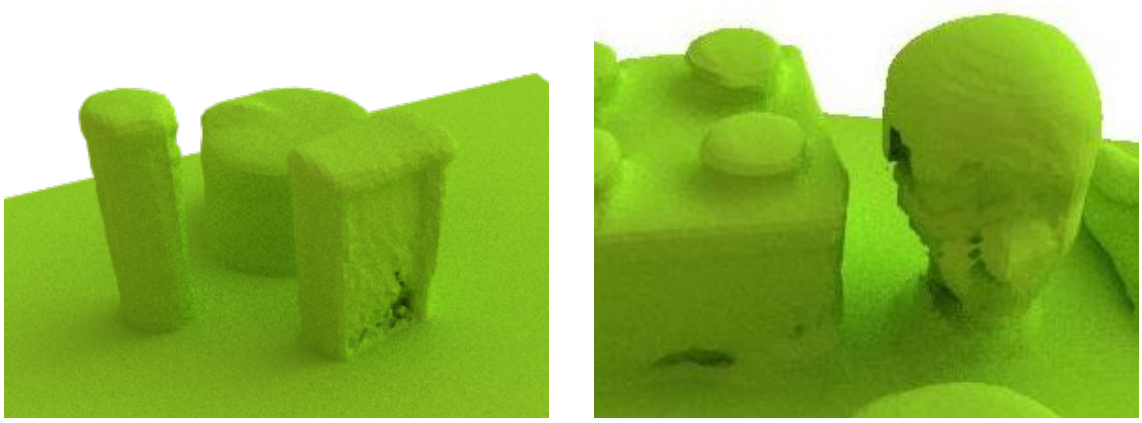


Figure 7.3: Proposed Mechanical Turk system to evaluate completions

object. We could therefore consider using the true/false success metric of 'did the robot manage to successfully pick up the target object without any collisions', using a path planned using the predicted occupancy grid. This could be performed in a robotic simulation system, such as OpenGRASP [104], as running experiments with real robots can be time consuming and costly, and it can be very difficult to accurately compare algorithms in real-world test scenarios.

⇛ For an image editing application we care less about accuracy compared to the ground truth and more about generating a plausible looking completion. One way to evaluate this would be to survey humans, using e.g. a method presented by [62] in order to see which completions look more plausible. Users on the internet, recruited using e.g. Amazon's Mechanical Turk, could compare potential completions to judge which are the most plausible. We present a mock-up of such a system in Figure 7.3. This type of comparison can be expensive to run, so we could instead consider a proxy measure which could be cheaply evaluated locally, designed to encode the perceptual plausibility of completions.



(a) (b)

Figure 7.4: Examples of where the ground truth volumes we have obtained appear to have missing data (a) and inconsistencies (b).

7.3 Is the ground truth good enough?

When collecting the tabletop dataset we found that the camera tracking sometimes performed poorly, leading to poor estimation of the relative transform between single frames and the world model. Together with the limited quality of the individual Kinect frames, this means that there are potential issues with the ground truth reconstruction of the scenes. Some errors which are visible to the human eye are shown in Figure 7.4. This potentially negatively effects our work in three ways:

1. Our test-time comparisons may in some cases be against a scene shape which is not representative of the true scene shape

2. Even if the global model is well constructed, individual depth images may not be well aligned relative to it. This means that for a specific test image, the ground truth grid may be misaligned by a small translation and rotation.

3. Our training data may suffer from the problems in items 1) and 2). This means that the models learned may make worse predictions than they might do if such problems did not exist with the training data.

In practice though, we found that all our scenes have reconstructed to a very plausible looking shape using the fusion implementation of [130]. Issues like those shown in

Figure 7.4 are rare and small compared to the overall shape gained. A more accurate registration could be achieved by using an offline registration algorithms e.g. [24], which tend to be more robust to problems such as areas with limited depth information, or a more recent real-time system (e.g. [173]) which have shown considerable improvements over standard fusion.

7.3.1 Training data used

The training and test data we used is limited to lab experiments. The next step in using this data would be to take the algorithms out of the lab and apply them to the real world. At this stage, it may be the case that certain strengths and limitations of each of the algorithms come in to play. It is likely that the most suitable algorithm for scaling up to real-world prediction may well be application specific.

7.4 Future work for solving the occupancy problem

In each chapter we have set out the work that we consider to be the most important for future exploration. Here, we present some future work areas common to all approaches.

Memory efficient voxel storage We made the decision to model the world as a grid of fixed-size voxels, following works such as [80]. Modelling the world as a grid is a very natural analogy to image pixels, and it makes storage and manipulation of shape data easy. The primary challenge with voxel representations of the world is the high memory cost. Storing a $5m \times 5m \times 3m$ room with a $(5mm)^3$ voxel grid would require 0.6GB, assuming one byte per voxel. However, the vast majority of voxels in such a scene are likely to be empty. Finding more efficient ways of storing the data, for example by using hierarchical occupancy trees, would be an important area of development to make the algorithms scalable to real-world training and testing.

Removing the reliance on the ground plane In our work we have assumed the presence of a ground plane on which our objects of interest sit. Training, testing and developing our algorithms on data which go beyond this restriction is an important next step for developing the completion algorithms for the real world.

(a) 'Large Dataset of Object Scans' [25] (b) 'ShapeNet' [19]

Figure 7.5: New datasets which could be modified to allow for use with our algorithms

An occupancy model using deep learning A recent avenue in computer vision and machine learning research has been deep learning using neural networks — see for example [42]. A deep learning representation of occupancy prediction could be used to map an input image (or an input grid, such as that proposed in Section 4.5) to a full occupancy grid. Our voxlets algorithm (Chapter 5) lends itself well to use with a deep learning framework. The structured Random Forest could be replaced by a convolutional neural network, enabling many of the benefits of deep learning to be exploited within our framework.

Using more data In our work so far, we have used our own synthetic and real datasets for training and test. Our algorithms have been designed to enable the use of limited amount of training data in order to learn its completions. Deep learning frameworks tend to be fairly 'data-hungry'; neural networks operating on 2D images are often trained on datasets of millions of examples [98], and the networks are able to exploit the information contained within these data.

For our work, we envisage that using larger datasets for training would allow for completions on a more diverse range of test scenes. The recently released 'Large Dataset of Object Scans' [25] could, with some modification, be used for our task. In addition, the synthetic ShapeNet dataset [19] could be use to build synthetic scenes with more detailed geometry than the data we generated. See Figure 7.5 for examples of these datasets.

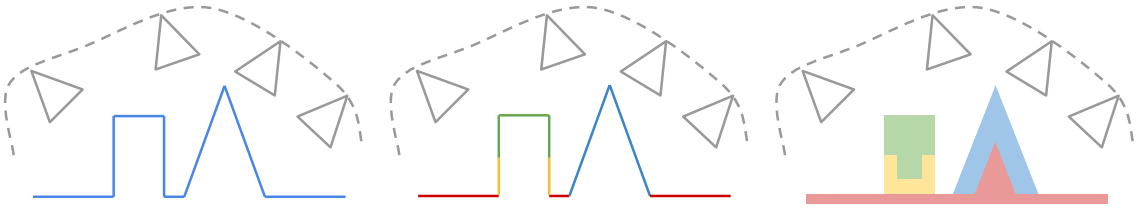
7.5 Future work extending the scope of the problem

In this thesis we have ultimately focussed solely on the problem of predicting the binary occupancy value of voxels in a scene from a single depth image. There are two areas of future work which could extend scope of the problem:

Per-voxel segmentation In addition to predicting the occupancy of each voxel in the scene, we would like to be able to segment the voxel grid into regions, each of which corresponds to a different, moveable object. This could be an object segmentation algorithm, or dense semantic labelling — see Figure 7.6. This would help a robot to answer the question: "what would this scene look like if I took this object away?". Generating training and test data for such a voxel segmentation problem is difficult, as we desire a labelling at each point in 3D space (in contrast to a labelling just on the surface of a mesh).

Utilising with robotic grasping Our test sets are strongly linked to robot grasping scenarios. An excellent demonstration of how computer vision can interface with a robotic arm for grasping is given in [11]. Refining our prediction algorithms with robotic grasping in mind would be a valuable contribution to knowledge. Issues that may need to be addressed could include speed of predictions and developing more suitable accuracy metrics (Section 7.2).

Multi-frame input for voxel occupancy prediction In this scenario, we consider extending the problem to predicting voxel occupancy given a sequence of depth images as input. As each depth image arrives, we should be able to use the additional information to improve our predictions. The datasets we introduced in this thesis could be used for this purpose, and our algorithms may be able to be adapted to this problem setup with relatively little adjustment. One interesting potential application of this extension would be to use the predicted completion as a prior for SLAM. As new data arrives, a next-best-view algorithm [128] could leverage the predictions made in order to suggest where the camera should be moved to in order to maximise the expected information gain for the voxel completion.



(a) In our datasets, we move the camera to view all the visible surfaces

(b) We could consider the problem of applying surface labels to a recovered mesh

(c) A more advanced alternative could be to reason about labels densely at every point in 3D

Figure 7.6: A concept for dense voxel semantic labelling

Appendices

Appendix A

Notation

The types of symbols used in this report to denote different mathematical types are shown in table A.1. This follows mathematical convention and is well described in appendix A of [129].

	Symbol type	**Examples**
Scalars	Letters and greek symbols	D, θ
Vectors and tuples	Bold lowercase letters	$\mathbf{v}$, $\mathbf{a}$
Matrices	Bold uppercase letters	$\mathbf{D}$, $\mathbf{A}$
Sets	Caligraphic letters	$\mathcal{D}$, $\mathcal{S}$
Sets of sets	Fraktur symbols	$\mathfrak{D}$, $\mathfrak{A}$

Table A.1: Types of symbol used in this thesis

$[\bullet]$ is the Iverson bracket, which evaluates to 1 if $\bullet$ evaluates to true and 0 otherwise.

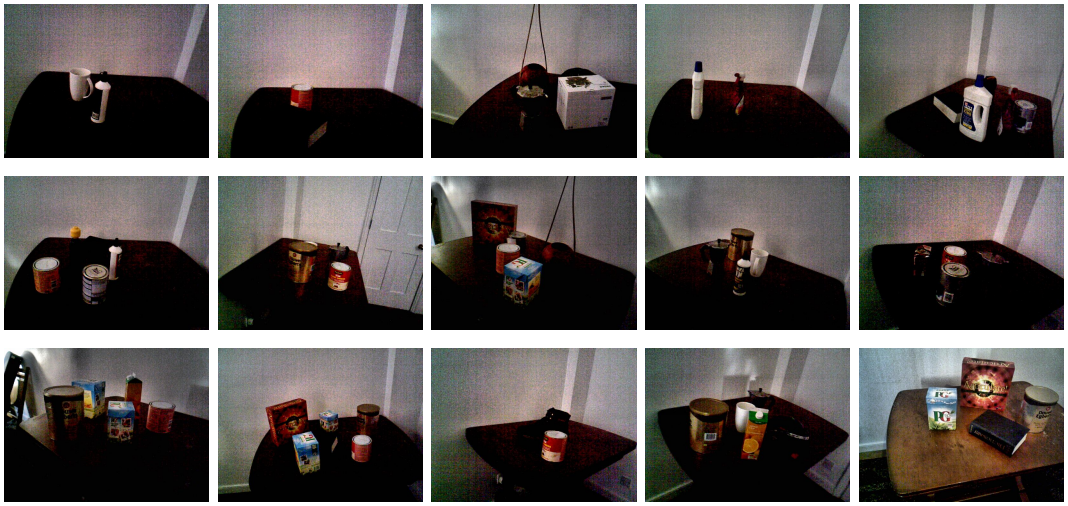
Appendix B

All images from the tabletop dataset

Here we present a single RGB image from of each training sequence. Each training sequence consists of several hundred images of the scene, which we fuse to get the ground truth volume. For our algorithm we selected four images from each sequence to be used as images in the training set.

Note that the images in the first half of the training set were captured under low light conditions, hence are darker in color than those in the second half.

B.1 Fold 1 (training)



B.2 Fold 2 (training)

B.3 Fold 3 (test)

Appendix C

RGBD features

In this appendix we set out how we compute the features used in our experiments on RGBD images. Each object feature is computed for a region $\mathcal{R} \subset \mathbf{D}$, while local features are computed for the points or pixels in a neighbourhood $\mathcal{N}(\mathbf{p})$ of a certain pixel $\mathbf{p}$.

In addition to its raw state as an image region, each $\mathcal{R}$ can be represented as a point cloud $\mathcal{P} = \{\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_N\}$, where each point $\mathbf{w} = (x, y, z)$ is a position in three-dimensional Cartesian coordinates. A point cloud can also be represented as an $N \times 3$ matrix $P = [\mathbf{w}_1^T, \mathbf{w}_2^T, \cdots, \mathbf{w}_N^T]^T$.

C.1 Bounding box size

The size of the bounding box of a region is computed from the region's projected point cloud $\mathcal{P}$. First, the principal directions $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ of P are found via eigenvalue decomposition. These are stacked in a 3×3 matrix $R = [\mathbf{e}_1^T, \mathbf{e}_2^T, \mathbf{e}_3^T]^T$. Next the points in P are transformed into their principal axes, creating a transformed point cloud $\hat{P}$:

$$\hat{P} = (RP^T)^T. \tag{C.1}$$

The bounding box size feature vector $\mathbf{x}_{\texttt{bounding\_box}}$ can then be found by find the range of points along each dimension of $\hat{P}$, i.e.

$$\mathbf{x}_{\texttt{bounding\_box}} = \begin{pmatrix} \max(\hat{P}_1) - \min(\hat{P}_1) \\ \max(\hat{P}_2) - \min(\hat{P}_2) \\ \max(\hat{P}_3) - \min(\hat{P}_3) \end{pmatrix}, \tag{C.2}$$

where $\hat{P}_i$ is the ith column of $\hat{P}$.

C.2 RGB mean

The value for the mean RGB is simply the arithmetic mean of each colour channel $\in \{\texttt{red}, \texttt{green}, \texttt{blue}\}$ present in the image region $\mathcal{R}$:

$$\mathbf{x}_{\texttt{rgb\_mean}} = \left(\frac{\sum_{i=1}^{N} r_i}{N}, \frac{\sum_{i=1}^{N} g_i}{N}, \frac{\sum_{i=1}^{N} b_i}{N} \right). \tag{C.3}$$

C.3 RGB histogram

The histogram we take over RGB values is a formed as a $n \times n \times n$ array X, where each dimension in X represents a colour channel $\in \{\texttt{red}, \texttt{green}, \texttt{blue}\}$. Each pixel $\mathbf{p}$ has colour represented as a vector $(r, g, b) \in [0, 1]^{1\times 3}$, which is mapped to a histogram bin via the transformation

$$(\psi^r, \psi^g, \psi^b) = \lfloor r(n-1) + 1, g(n-1) + 1, b(n-1) + 1 \rfloor. \tag{C.4}$$

Each bin in X accumulates the pixels as so:

$$X(i, j, k) = \sum_{q=1}^{N} \Big[(\psi^r = i) \wedge (\psi^g = j) \wedge (\psi^b = k) \Big], \tag{C.5}$$

where $[\bullet]$ is the Iverson bracket.

Finally X is reshaped into a $1 \times n^3$ vector $\mathbf{x}_{\texttt{rgb\_histogram}}$. In our work we set $n = 4$. In future work, it may be worth considering forming a proper dictionary of visual words rather than a histogram with fixed bin widths.

C.4 Shape distributions

Shape distributions are a feature proposed by Osada et al. [121] that aim to capture the overall geometry of an object 'in an invariant way'. They work by sampling multiple subsets of points from the input object, and computing a histogram over metrics computed from the subsets of points. They propose five variants of the distribution, each using a different measurement:

A3: The angle between three random points.

D1: The distance between a fixed point (e.g. the centroid) and a random point.

D2: The distance between two random points.

D3: The square root of the area of the triangle between three random points.

D4: The cube root of the volume of the tetrahedron between four random points.

In their work, Osada et al. find that the D2 distribution performs the most effectively. This is what we use in our work. We form a histogram over the L_2 distance between $5,000$ randomly sampled pairs of points in $\mathcal{P}$. We specify a non-linear, quasi-logarithmic distribution over edges for the bins to try to capture detail at hierarchical levels of detail. The edges we choose take the values

$$[0, 0.01, 0.02, \cdots, 0.05, 0.075, 0.1, \cdots, 0.5, 0.6, 0.7, \cdots, 1.5]. \tag{C.6}$$

C.5 Histogram of Gradients (HOG)

Histograms of Orientated Gradients have most famously been used for pedestrian detection [34]. We use a simplified version of their implementation, and our methodology is as follows:

First the whole image $\mathbf{D}$ is converted to a greyscale image $\mathbf{D}_g$. This is then separately convoluted with the filters $[-1, 0, 1]$ and $[-1, 0, 1]^T$ to create images $\frac{\partial \mathbf{D}_g}{\partial x}$ and $\frac{\partial \mathbf{D}_g}{\partial y}$ respec-

tively. The intensity I and orientation θ images are then computed using the formulae:

$$I = \sqrt{\frac{\partial \mathbf{D}_g}{\partial x}^2 + \frac{\partial \mathbf{D}_g}{\partial y}^2} \tag{C.7}$$

$$\theta = \text{atan2}\left(\frac{\partial \mathbf{D}_g}{\partial x}, \frac{\partial \mathbf{D}_g}{\partial y}\right) \tag{C.8}$$

The final descriptor for each is a $1 \times n$ histogram $\mathbf{x}_{\texttt{hog}}$ over the region's θ values, where each value is weighted by the corresponding I value:

$$\mathbf{x}_{\texttt{hog}}(k) = \sum_{i=1}^{N} \left[\left\lfloor \frac{\theta_i(n-1)}{2\pi} \right\rfloor + 1 = k \right] \cdot I_i \tag{C.9}$$

In the original paper which introduced histograms of gradients the image gamma and colour was normalised as a pre-processing step. Furthermore, the gradients were computed over a grid of overlapping cells and the combination of all feature vectors are used in the overall feature vector. In our simplified version of HOG we set $n = 9$.

C.6 Spin images

Spin images are a descriptor proposed by Johnson et al. [85] for locating objects in cluttered 3D scenes with multiple occlusions; they have since become a staple feature in 3D classification work. The spin image S at point $\mathbf{p} = (x, y, z)$ can be thought of as a 2D array formed by orientating a grid with the normal $\mathbf{n}_\mathbf{p}$, and rotating it through a full circle. Each cell in the grid array 'accumulates' each point that falls into it, eventually storing the total number of points.

We use two parameters to affect the image created:

Support distance (W_α, W_β) is the size of the image created in each of its two dimensions; this corresponds to setting the size of the neighbourhood used. By adjusting the support distance, spin images can be converted from a global to a local descriptor, and from a less to a more discriminating feature [38].

Raster resolution (R_α, R_β) is the number of cells in each dimension of the 2D histogram. This affects the level of detail captured.

For the small, local spin images produced in this report, the support distance and the raster resolution were the only parameters varied. The support angle was kept at the maximum of π, and the attenuation function was uniform with distance from $\mathbf{p}$.

The algorithm we use for computing the spin images is as follows:

1. Given a query point $\mathbf{p} \in \mathcal{P}$, and its normal $\mathbf{n_p}$, each other point in $\mathbf{x} \in \mathcal{P} : \mathbf{x} \neq \mathbf{p}$ is transformed into (α, β) coordinates:

$$\begin{aligned} \alpha &= \sqrt{|\mathbf{x}-\mathbf{p}|^2 - (\mathbf{n}\cdot(\mathbf{x}-\mathbf{p}))^2} \\ \beta &= \mathbf{n}\cdot(\mathbf{x}-\mathbf{p}) \end{aligned}$$

2. A set $\mathcal{S}$ is formed of all the points that fall within a pre-defined support distance (W_α, W_β):

$$\mathcal{S} = \{(\alpha_j, \beta_j)\} \quad | \quad \alpha_j < W_\alpha, \quad \beta_j < W_\beta$$

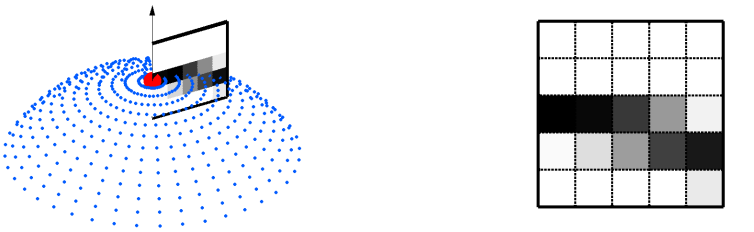
3. The points in $\mathcal{S}$ are each assigned to a bin $S_{i,j}$ in the $R_\alpha \times R_\beta$ 2D histogram $S_\mathbf{p}$. The index of the bin $S_{i,j}$ each point is assigned to is given as:

$$i = \left\lfloor \frac{\alpha R_\alpha}{W_\alpha} \right\rfloor, \quad j = \left\lfloor \frac{\beta R_\beta}{W_\beta} + \frac{R_\beta}{2} \right\rfloor. \tag{C.10}$$

4. Each bin $S_{i,j}$ is then equal to the number of points with coordinates (i, j).

As the array is orientated with the surface normal at $\mathbf{p}$, and the rotational position of points around this normal are ignored, spin images are invariant to both the position and the orientation of the object.

We computed 100 spin images for each $\mathcal{R}$, with support distance $W_\alpha = W_\beta = 0.05m$ and raster resolution $R_\alpha = R_\beta = 5$. We made a dictionary by clustering together a representative sample of spin images into 50 clusters using k-means. Our final feature vector $\mathbf{x}_{\texttt{spin\_image}}$ is then a 1×50 vector where each element represents the number of spin images in the region which were assigned to that dictionary entry via the nearest neighbour approach.



(a) Spin image creation in context

(b) Example 5×5 spin image

Figure C.1: Illustration of spin image creation. The 5×5 raster image S is spun around the axis formed by the normal at the red point $\mathbf{p}$, and each cell in S accumulates the points that fall into it.

Appendix D

The adjusted Rand index

In this appendix we set out the formula for the computation of the adjusted Rand index (ARI) [78], which is used to evaluate the accuracy of clustering.

Given a ground truth labelling $\{g_1, \ldots, g_M\}$ and our inferred labels $\{l_1, \ldots, l_M\}$, each pair of objects $\{(\mathbf{w}_i, \mathbf{w}_j) | 1 \leq i < j \leq M\}$ is assigned to one of four sets. The cardinalities of these sets, a, b, c and d, are described in Table D.1.

	Ground truth	
Inferred label	Pair in same group	Pair in different groups
Pair in same group	True positive a pairs	False positive b pairs
Pair in different groups	False negative c pairs	True negative d pairs

Table D.1: The four sets used for calculation of the ARI

The ARI is then computed as

$$ARI = \frac{\binom{n}{2}(a+d) - [(a+b)(a+c)+(c+d)(b+d)]}{\binom{n}{2}^2 - [(a+b)(a+c)+(c+d)(b+d)]} \in [-1, 1]. \tag{D.1}$$

Although the ARI can theoretically take any value in the interval $[-1, 1]$, in practice it only falls below zero in exceptional circumstances. For that reason we only display graphs plotting the index showing the interval $[0, 1]$. Some examples of adjusted Rand indexes for different classifications are shown in Figure D.1.

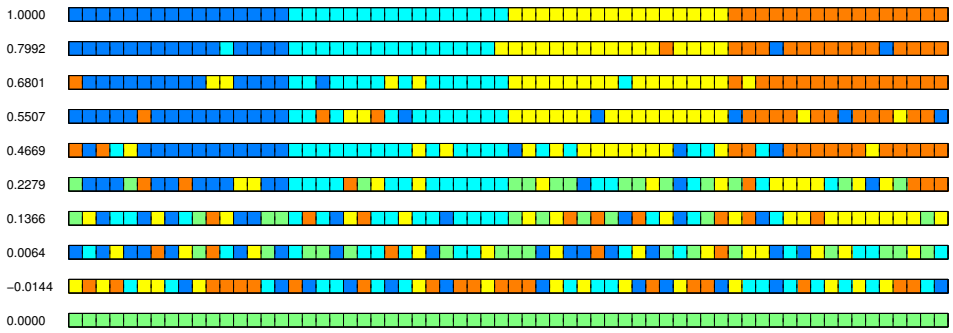


Figure D.1: Examples of ARI scores for different levels of inaccuracy. Each row represents a hypothetical clustering of data points, where each colour represents a different cluster. In this example we assume the top row to represent the perfect classification, which by definition has an ARI of 1.0. This is corrupted with different levels of noise, and associated ARIs are shown to the left of each row.

Bibliography

[1] A. Aldoma and A. Richtsfeld. The willow garage object recognition challenge. `http://www.acin.tuwien.ac.at/forschung/v4r/mitarbeiterprojekte/willow/`, 2012. Accessed: 01/06/2015.

[2] A. Aldoma, F. Tombari, L. di Stefano, and M. Vincze. A global hypotheses verification method for 3D object recognition. In *European Conference on Computer Vision (ECCV)*, pages 511–524, 2012.

[3] S. Bagon and M. Galun. Large scale correlation clustering optimization. *arXiv:1112.2903*, abs/1112.2903, 2011.

[4] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2002.

[5] J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society*, pages 259–302, 1986.

[6] P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *Pattern Analysis and Machine Intelligence (PAMI)*, pages 586–606, 1992.

[7] I. Biederman. Recognition-by-components: A theory of human image understanding. *Psychological Review*, 94(2):115, 1987.

[8] I. Biederman. *Human Image Understanding*. PN, 1989.

[9] M. Blane, Z. B. Lei, and D. B. Cooper. The 3L algorithm for fitting implicit polynomial curves and surfaces to data. *Pattern Analysis and Machine Intelligence (PAMI)*, 22(3):298–313, 2000.

[10] D. M. Blei, A. Ng, and M. I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[11] J. Bohg. *Multi-Modal Scene Understanding for Robotic Grasping*. PhD thesis, Kungliga Tekniska högskolan School of Computer Science and Communication, 2011.

[12] H. Bostrom. Estimating class probabilities in Random Forests. In *Conference on Machine Learning and Applications*, pages 211–216, 2007.

[13] L. Bourdev and J. Malik. Poselets: Body part detectors trained using 3D human pose annotations. In *International Conference on Computer Vision (ICCV)*, pages 1365–1372, 2009.

[14] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via Graph Cuts. In *International Conference on Computer Vision (ICCV)*, pages 1222–1239, 1999.

[15] T. P. Breckon and R. B. Fisher. Amodal volume completion: 3D visual completion. *Computer Vision and Image Understanding*, 99(3):499–526, 2005.

[16] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[17] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla. Segmentation and recognition using structure from motion point clouds. In *European Conference on Computer Vision (ECCV)*, pages 44–57, 2008.

[18] J. Carreira and C. Sminchisescu. Constrained parametric min-cuts for automatic object segmentation. In *Computer Vision and Pattern Recognition (CVPR)*, pages 3241–3248, 2010.

[19] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. *arXiv:1512.03012*, 2015.

[20] H. J. Chang, C. S. G. Lee, Y.-H. Lu, and Y. C. Hu. P-SLAM: Simultaneous localization and mapping with environmental-structure prediction. *Transactions on Robotics*, 23(2):281–293, 2007.

[21] D. Chekhlov, A. P. Gee, A. Calway, and W. Mayol-Cuevas. Ninja on a plane: Automatic discovery of physical planes for augmented reality using visual SLAM. In *ACM International Symposium on Mixed and Augmented Reality*, pages 1–4, 2007.

[22] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. In *International Conference on Robotics and Automation (ICRA)*, pages 2724–2729, 1991.

[23] F. Chierichetti, N. Dalvi, and R. Kumar. Correlation clustering in MapReduce. In *International Conference on Knowledge Discovery and Data Mining*, pages 641–650, 2014.

[24] S. Choi, Q.-Y. Zhou, and V. Koltun. Robust reconstruction of indoor scenes. In *Computer Vision and Pattern Recognition (CVPR)*, pages 5556–5565, 2015.

[25] S. Choi, Q.-Y. Zhou, S. Miller, and V. Koltun. A large dataset of object scans. *arXiv:1602.02481*, 2016.

[26] W.-S. Chu, C.-P. Chen, and C.-S. Chen. MOMI-cosegmentation: Simultaneous segmentation of multiple objects among multiple images. In *Asian Conference on Computer Vision (ACCV)*, pages 355–368, 2010.

[27] T. T. Cocias, F. Moldoveanu, and S. M. Grigorescu. Generic fitted primitives (GFP): Towards full object volumetric reconstruction for service robotics. In *Computer Graphics, Visualization and Computer Vision*, 2013.

[28] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. In *European Conference on Computer Vision (ECCV)*, pages 681–685, 1998.

[29] A. Criminisi, P. Perez, and K. Toyama. Object removal by exemplar-based inpainting. In *Computer Vision and Pattern Recognition (CVPR)*, pages II–721, 2003.

[30] A. Criminisi and J. Shotton. *Decision forests for computer vision and medical image analysis*. Springer Science, 2013.

[31] A. Criminisi, J. Shotton, and S. Bucciarelli. Decision forests with long-range spatial context for organ localization in CT volumes. In *MICCAI Workshop on Probabilistic Models for Medical Image Analysis*, pages 69–80, 2009.

[32] G. Csurka, D. Larlus, and F. Perronnin. What is a good evaluation measure for semantic segmentation? In *British Machine Vision Conference (BMVC)*, 2013.

[33] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Computer graphics and interactive techniques*, pages 303–312, 1996.

[34] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition (CVPR)*, pages 886–893, 2005.

[35] J. Davis, S. R. Marschner, M. Garr, and M. Levoy. Filling holes in complex surfaces using volumetric diffusion. In *3DPVT*, pages 428–441, 2002.

[36] T. Deselaers, B. Alexe, and V. Ferrari. Localizing objects while learning their appearance. In *European Conference on Computer Vision (ECCV)*, pages 452–466, 2010.

[37] S. J. Dickinson, R. Bergevin, I. Biederman, J.-O. Eklundh, R. Munck-Fairwood, A. K. Jain, and A. Pentland. Panel report: The potential of geons for generic 3-D recognition. *Image and Vision Computing*, 15(4):277–292, 1997.

[38] H. Q. Dinh and S. Kropac. Multi-resolution spin-images. In *Computer Vision and Pattern Recognition (CVPR)*, pages 863–870, 2006.

[39] P. Dollár and C. L. Zitnick. Structured forests for fast edge detection. In *International Conference on Computer Vision (ICCV)*, pages 1841–1848, 2013.

[40] P. Dollár and C. L. Zitnick. Fast edge detection using structured forests. *Pattern Analysis and Machine Intelligence (PAMI)*, 37(8):1558–1570, 2015.

[41] B. Drost and S. Ilic. 3D object detection and localization using multimodal point pair features. In *3DIMPVT*, pages 9–16, 2012.

[42] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Neural Information Processing (NIPS)*, pages 2366–2374, 2014.

[43] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[44] F. Endres, C. Plagemann, C. Stachniss, and W. Burgard. Unsupervised discovery of object classes from range data using latent Dirichlet allocation. In *Robotics Science and Systems (RSS)*, pages 113–120, 2009.

[45] S. M. A. Eslami, N. Heess, and J. Winn. The Shape Boltzmann Machine: a strong model of object shape. In *Computer Vision and Pattern Recognition (CVPR)*, pages 155–176, 2012.

[46] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[47] D. Ferguson, M. Likhachev, and A. Stentz. A guide to heuristic-based path planning. In *International Conference on Automated Planning & Scheduling*, pages 9–18, 2005.

[48] M. Firman. List of RGBD datasets. `http://www0.cs.ucl.ac.uk/staff/M.Firman/RGBDdatasets/`, 2015. Accessed: 13/06/2015.

[49] M. Firman. RGBD datasets: Past, present and future. In *Computer Vision and Pattern Recognition (CVPR)Workshop on Large Scale 3D Data: Acquisition, Modelling and Analysis*, 2016. Preprinted as arXiv:1604.00999.

[50] M. Firman and S. Julier. The effect of noise in unsupervised range data classification. In *Intelligent Robots and Systems (IROS)*, page 234, 2011.

[51] M. Fisher, D. Ritchie, M. Savva, T. Funkhouser, and P. Hanrahan. Example-based synthesis of 3D object arrangements. *ACM Transactions on Graphics*, 31(6):135, 2012.

[52] G. Flitton, T. P. Breckon, and N. Megherbi. A comparison of 3D interest point descriptors with application to airport baggage object detection in complex CT imagery. *Pattern Recognition*, 46(9):2420–2436, 2011.

[53] D. F. Fouhey, A. Gupta, and M. Hebert. Data-driven 3D primitives for single image understanding. In *International Conference on Computer Vision (ICCV)*, pages 3392–3399, 2013.

[54] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[55] H. Fu, D. Xu, S. Lin, and J. Liu. Object-based RGBD image co-segmentation with mutex constraint. In *Computer Vision and Pattern Recognition (CVPR)*, pages 4428–4436, 2015.

[56] K. Fukunaga and L. D. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, 1975.

[57] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski. Manhattan-world stereo. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1422–1429, 2009.

[58] N. Gelfand, N. J. Mitra, L. J. Guibas, and H. Pottmann. Robust global registration. In *European Symposium on Geometry Processing*, page 5, 2005.

[59] R. Girshick, J. Shotton, P. Kohli, A. Criminisi, and A. Fitzgibbon. Efficient regression of general-activity human poses from depth images. In *International Conference on Computer Vision (ICCV)*, pages 415–422, 2011.

[60] E. B. Goldstein. *Perceiving Objects and Scenes*. Cengage Learning, 2009.

[61] D. Gossow, D. Weikersdorfer, and M. Beetz. Distinctive texture features from perspective-invariant keypoints. In *International Conference on Pattern Recognition*, pages 2764–2767, 2012.

[62] M. Gryka. imcompadre. `http://http://www.imcompadre.com`, 2015. Accessed: 01/06/2015.

[63] R. Guo and D. Hoiem. Support surface prediction in indoor scenes. In *International Conference on Computer Vision (ICCV)*, pages 2144–2151, 2013.

[64] A. Gupta, S. Satkin, A. A. Efros, and M. Hebert. From 3D scene geometry to human workspace. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1961–1968, 2011.

[65] N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[66] R. Hammer, T. Hertz, S. Hochstein, and D. Weinshall. Category learning from equivalence constraints. *Cognitive Processing*, 10(3):211–232, 2009.

[67] A. Handa, T. Whelan, J. McDonald, and A. J. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *International Conference on Robotics and Automation (ICRA)*, pages 1524–1531, 2014.

[68] G. Harary and A. Tal. Context-based coherent surface completion. *ACM Transactions on Graphics*, 33(1):5, 2013.

[69] J. Hays and A. A. Efros. Scene completion using millions of photographs. *ACM Transactions on Graphics*, 26(3):4, 2007.

[70] X. He, R. S. Zemel, and M. A. Carreira-Perpiñán. Multiscale conditional random fields for image labeling. In *Computer Vision and Pattern Recognition (CVPR)*, pages II–695, 2004.

[71] V. Hedau, D. Hoiem, and D. Forsyth. Recovering free space of indoor scenes from a single image. In *Computer Vision and Pattern Recognition (CVPR)*, pages 2807–2814, 2012.

[72] E. Herbst, X. Ren, and D. Fox. RGB-D object discovery via multi-scene analysis. In *Intelligent Robots and Systems (IROS)*, pages 4850–4856, 2011.

[73] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *International Conference on Computer Vision (ICCV)*, pages 858–865, 2011.

[74] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab. Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes. In *Asian Conference on Computer Vision (ACCV)*, pages 548–562, 2012.

[75] D. Holz, S. Holzer, R. B. Rusu, and S. Behnke. Real-time plane segmentation using RGB-D cameras. In *RoboCup 2011: robot soccer world cup XV*, pages 306–317. Springer, 2011.

[76] A. Hornung, M. Phillips, E. G. Jones, M. Bennewitz, M. Likhachev, and S. Chitta. Navigation in three-dimensional cluttered environments for mobile manipulation. In *International Conference on Robotics and Automation (ICRA)*, pages 423–429, 2012.

[77] D. F. Huber and M. Hebert. Fully automatic registration of multiple 3d data sets. *Image and Vision Computing*, 21(7):637–650, 2003.

[78] L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.

[79] Ikea. Ikea facts & figures. `http://www.ikea.com/ms/en_GB/about_ikea/facts_and_figures/`, 2015. Accessed: 20/06/2015.

[80] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568, 2011.

[81] A. Janoch, S. Karayev, Y. Jia, J. T. Barron, M. Fritz, K. Saenko, and T. Darrell. A category-level 3-D object dataset: Putting the Kinect to work. In *International Conference on Computer Vision (ICCV)Workshop on Consumer Depth Cameras in Computer Vision*, pages 141–165, 2011.

[82] Z. Jia, A. Gallagher, A. Saxena, and T. Chen. 3D-based reasoning with blocks, support, and stability. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2013.

[83] E. Johns, S. Leutenegger, and A. J. Davison. Pairwise decomposition of image sequences for active multi-view recognition. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[84] A. Johnson and M. Hebert. Surface matching for object recognition in complex 3-D scenes. *Image and Vision Computing*, 1998.

[85] A. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *Pattern Analysis and Machine Intelligence (PAMI)*, 21(5):433–449, 1999.

[86] M. Johnson-Roberson and J. Bohg. Attention-based active 3D point cloud segmentation. In *Intelligent Robots and Systems (IROS)*, pages 1165–1170, 2010.

[87] T. Ju. Fixing geometric errors on polygonal models: a survey. *Journal of Computer Science and Technology*, 24(1):19–29, 2009.

[88] H. Kang, M. Hebert, and T. Kanade. Discovering object instances from scenes of daily living. In *International Conference on Computer Vision (ICCV)*, 2011.

[89] A. Karpathy, S. Miller, and L. Fei-Fei. Object discovery in 3D scenes via shape analysis. In *International Conference on Robotics and Automation (ICRA)*, pages 2088–2095, 2013.

[90] N. Kholgade, T. Simon, A. Efros, and Y. Sheikh. 3D object manipulation in a single photograph using stock 3D models. *ACM Transactions on Graphics*, 33(4):127, 2014.

[91] B.-s. Kim, P. Kohli, and S. Savarese. 3D scene understanding by voxel-CRF. In *International Conference on Computer Vision (ICCV)*, pages 1425–1432, 2013.

[92] G. Kim, E. Xing, and L. Fei-Fei. Distributed cosegmentation via submodular optimization on anisotropic diffusion. In *International Conference on Computer Vision (ICCV)*, pages 169–176, 2011.

[93] J. Kim and K. Grauman. Shape sharing for object segmentation. In *European Conference on Computer Vision (ECCV)*, pages 444–458, 2012.

[94] Y. M. Kim, N. J. Mitra, D.-M. Yan, and L. Guibas. Acquiring 3D indoor environments with variability and repetition. *ACM Transactions on Graphics*, 31(6):138, 2012.

[95] P. Kontschieder, S. R. Bulo, H. Bischof, and M. Pelillo. Structured class-labels in random forests for semantic image labelling. In *International Conference on Computer Vision (ICCV)*, pages 2190–2197, 2011.

[96] H. S. Koppula, A. Anand, and T. Joachims. Labeling 3D scenes for personal assistant robots. In *Robotics Science and Systems (RSS)Workshop on RGB-D Cameras*, 2011.

[97] H. S. Koppula, A. Anand, T. Joachims, and A. Saxena. Semantic labeling of 3D point clouds for indoor scenes. In *Neural Information Processing (NIPS)*, pages 244–252, 2011.

[98] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing (NIPS)*, pages 1097–1105, 2012.

[99] O. Kroemer, H. B. Amor, M. Ewerton, and J. Peters. Point cloud completion using extrusions. In *International Conference on Humanoid Robots (HUMANOIDS)*, pages 680–685, 2012.

[100] K. Lai, L. Bo, and D. Fox. Unsupervised feature learning for 3D scene labeling. In *International Conference on Robotics and Automation (ICRA)*, pages 3050–3057, 2014.

[101] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view RGB-D object dataset. In *International Conference on Robotics and Automation (ICRA)*, pages 1817–1824, 2011.

[102] A. J. Law and D. G. Aliaga. Single viewpoint model completion of symmetric objects for digital inspection. *Computer Vision and Image Understanding*, 115(5):603–610, 2010.

[103] T. Leelasawassuk and W. Mayol-Cuevas. 3D from looking: Using wearable gaze tracking for hands-free and feedback-free object modelling. In *International Symposium on Wearable Computers*, pages 105–112, 2013.

[104] B. León, S. Ulbrich, R. Diankov, G. Puche, M. Przybylski, A. Morales, T. Asfour, S. Moisio, J. Bohg, J. Kuffner, et al. Opengrasp: a toolkit for robot grasping simulation. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 109–120, 2010.

[105] D. Liu and T. Chen. Unsupervised image categorization and object localization using topic models and correspondences between images. In *International Conference on Computer Vision (ICCV)*, pages 1–7, 2007.

[106] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM Transactions on Graphics*, pages 347–353, 1987.

[107] O. Mac Aodha, N. D. Campbell, A. Nair, and G. J. Brostow. Patch based synthesis for single depth image super-resolution. In *European Conference on Computer Vision (ECCV)*, pages 71–84, 2012.

[108] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[109] C. B. Madsen and R. Laursen. Image relighting: Getting the sun to set in an image taken at Noon. In *3th Danish Conference on Pattern Recognition and Image Analysis, DSAGM*, pages 13–20, 2004.

[110] J. Mason, B. Marthi, and R. Parr. Object disappearance for object discovery. In *Intelligent Robots and Systems (IROS)*, pages 2836–2843, 2012.

[111] O. Mattausch, D. Panozzo, C. Mura, O. Sorkine-Hornung, and R. Pajarola. Object detection and classification from large-scale cluttered indoor scans. In *Eurographics*, pages 11–21, 2014.

[112] S. Meister, S. Izadi, P. Kohli, M. Hämmerle, C. Rother, and D. Kondermann. When can we use KinectFusion for ground truth acquisition? In *Intelligent Robots and Systems (IROS) Workshop on Color-Depth Camera Fusion in Robotics*, 2012.

[113] A. S. Mian, M. Bennamoun, and R. Owens. Three-dimensional model-based object recognition and segmentation in cluttered scenes. *Pattern Analysis and Machine Intelligence (PAMI)*, 28(10):1584–1601, 2006.

[114] A. K. Mishra, A. Shrivastava, and Y. Aloimonos. Segmenting "simple" objects using RGB-D. In *International Conference on Robotics and Automation (ICRA)*, pages 4406–4413, 2012.

[115] U. Mohammed, S. J. Prince, and J. Kautz. Visio-lization: Generating novel facial images. *ACM Transactions on Graphics*, 28(3):57, 2009.

[116] A. Montillo, J. Shotton, J. Winn, J. E. Iglesias, D. Metaxas, and A. Criminisi. Entangled decision forests and their application for semantic segmentation of CT images. In *Information Processing in Medical Imaging*, pages 184–196, 2011.

[117] F. Moosmann and M. Sauerland. Unsupervised discovery of object classes in 3D outdoor scenarios. In *International Conference on Computer Vision (ICCV) Workshops*, pages 1038–1044, 2011.

[118] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Neural Information Processing (NIPS)*, pages 849–856, 2001.

[119] S. Nowozin and C. H. Lampert. Structured learning and prediction in computer vision. *Foundations and Trends in Computer Graphics and Vision*, 6(3–4):185–365, 2011.

[120] E. Olson, M. Walter, S. Teller, and J. Leonard. Single-cluster spectral graph partitioning for robotics applications. In *Robotics Science and Systems (RSS)*, pages 265–272, 2005.

[121] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. Matching 3D models with shape distributions. In *Shape Modeling International*, pages 154–166, 2001.

[122] M. R. Oswald, E. Töppe, and D. Cremers. Fast and globally optimal single view reconstruction of curved objects. In *Computer Vision and Pattern Recognition (CVPR)*, pages 534–541, 2012.

[123] D. Pelleg and A. Moore. X-means: Extending K-means with efficient estimation of the number of clusters. In *International Conference on Machine Learning (ICML)*, 2000.

[124] J. Philbin and J. Sivic. Geometric LDA: A generative model for particular object discovery. In *British Machine Vision Conference (BMVC)*, pages 1–10, 2008.

[125] C. Plagemann, S. Mischke, S. Prentice, K. Kersting, N. Roy, and W. Burgard. Learning predictive terrain models for legged robot locomotion. In *Intelligent Robots and Systems (IROS)*, pages 3545–3552, 2008.

[126] J. Podolak and S. Rusinkiewicz. Atomic volumes for mesh completion. In *Symposium on Geometry Processing*, pages 33–41, 2005.

[127] F. Pomerleau, S. Magnenat, F. Colas, M. Liu, and R. Siegwart. Tracking a depth camera: Parameter exploration for fast ICP. In *Intelligent Robots and Systems (IROS)*, pages 3824–3829, 2011.

[128] C. Potthast and G. S. Sukhatme. A probabilistic framework for next best view estimation in a cluttered environment. *Visual Communication and Image Representation*, 25(1):148–164, 2014.

[129] S. Prince. *Computer Vision Models*. Cambridge University Press, 2012.

[130] V. A. Prisacariu, O. Kähler, M. M. Cheng, C. Y. Ren, J. Valentin, P. H. Torr, I. D. Reid, and D. W. Murray. A framework for the volumetric integration of depth images. *arXiv:1410.0925*, 2014.

[131] V. A. Prisacariu and I. Reid. Shared shape spaces. In *International Conference on Computer Vision (ICCV)*, pages 2587–2594, 2011.

[132] C. Rennie, R. Shome, K. E. Bekris, and A. F. D. Souza. A dataset for improved RGBD-based object detection and pose estimation for warehouse pick-and-place. *IEEE Robotics and Automation Letters*, 2016.

[133] A. Richtsfeld, T. Mörwald, J. Prankl, M. Zillich, and M. Vincze. Segmentation of unknown objects in indoor environments. In *Intelligent Robots and Systems (IROS)*, pages 4791–4796, 2012.

[134] J. Rock, T. Gupta, J. Thorsen, J. Gwak, D. Shin, and D. Hoiem. Completing 3D object shape from one depth image. In *Computer Vision and Pattern Recognition (CVPR)*, pages 2484–2493, 2015.

[135] R. Rothe, M. Guillaumin, and L. V. Gool. Non-maximum suppression for object detection by passing messages between windows. In *Asian Conference on Computer Vision (ACCV)*, 2014.

[136] C. Rother, V. Kolmogorov, and A. Blake. "GrabCut": interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics*, 23(3):309–314, 2004.

[137] C. Rother, V. Kolmogorov, V. Lempitsky, and M. Szummer. Optimizing binary MRFs via extended roof duality. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2007.

[138] C. Rother, T. Minka, A. Blake, and V. Kolmogorov. Cosegmentation of image pairs by histogram matching–incorporating a global constraint into MRFs. In *Computer Vision and Pattern Recognition (CVPR)*, pages 993–1000, 2006.

[139] J. C. Rubio, J. Serrat, A. Lopez, and N. Paragios. Unsupervised co-segmentation through region matching. In *Computer Vision and Pattern Recognition (CVPR)*, pages 749–756, 2012.

[140] M. Ruhnke and B. Steder. Unsupervised learning of 3D object models from partial views. In *International Conference on Robotics and Automation (ICRA)*, pages 801–806, 2009.

[141] B. C. Russell, A. A. Efros, J. Sivic, W. T. Freeman, and A. Zisserman. Using multiple segmentations to discover objects and their extent in image collections. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1605–1614, 2006.

[142] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. J. Kelly, and A. J. Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1352–1359, 2013.

[143] S. Salti, F. Tombari, and L. D. Stefano. SHOT: Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*, 125:251–264, 2014.

[144] R. Schnabel, P. Degener, and R. Klein. Completion and reconstruction with primitive shapes. In *Computer Graphics Forum*, pages 503–512, 2009.

[145] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Computer Vision and Pattern Recognition (CVPR)*, pages 519–528, 2006.

[146] T. Shao, A. Monszpart, Y. Zheng, B. Koo, W. Xu, K. Zhou, and N. J. Mitra. Imagining the unseen: Stability-based cuboid arrangement for scene understanding. *ACM Transactions on Graphics*, 33(6):209–1, 2014.

[147] C.-H. Shen, H. Fu, K. Chen, and S.-M. Hu. Structure recovery by part assembly. *ACM Transactions on Graphics*, 31(6):180, 2012.

[148] J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence (PAMI)*, 22(8):888–905, 1997.

[149] J. Shin, R. Triebel, and R. Siegwart. Unsupervised discovery of repetitive objects. In *International Conference on Robotics and Automation (ICRA)*, pages 5041–5046, 2010.

[150] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *Computer Vision and Pattern Recognition (CVPR)*, pages 116–124, 2011.

[151] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon. Scene coordinate regression forests for camera relocalization in RGB-D images. In *Computer Vision and Pattern Recognition (CVPR)*, pages 2930–2937, 2013.

[152] N. Silberman and R. Fergus. Indoor scene segmentation using a structured light sensor. In *International Conference on Computer Vision (ICCV)Workshops*, pages 601–608, 2011.

[153] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from RGBD images. In *European Conference on Computer Vision (ECCV)*, pages 746–760, 2012.

[154] N. Silberman, L. Shapira, R. Gal, and P. Kohli. A contour completion model for augmenting surface reconstructions. In *European Conference on Computer Vision (ECCV)*, pages 488–503, 2014.

[155] A. Singh, J. Sha, K. Narayan, T. Achim, and P. Abbeel. A large-scale 3D database of object instances. In *International Conference on Robotics and Automation (ICRA)*, pages 509–516, 2014.

[156] S. Song, S. P. Lichtenberg, and J. Xiao. SUN RGB-D: A RGB-D scene understanding benchmark suite. In *Computer Vision and Pattern Recognition (CVPR)*, pages 567–576, 2015.

[157] S. Song and J. Xiao. Tracking revisited using RGBD camera: Unified benchmark and baselines. In *International Conference on Computer Vision (ICCV)*, pages 233–240, 2013.

[158] A. Stentz. Optimal and efficient path planning for partially-known environments. In *International Conference on Robotics and Automation (ICRA)*, pages 3310–3317, 1994.

[159] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *Intelligent Robots and Systems (IROS)*, pages 573–580, 2012.

[160] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

[161] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei. Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 2006.

[162] S. Thrun and B. Wegbreit. Shape from symmetry. In *International Conference on Computer Vision (ICCV)*, pages 1824–1831, 2005.

[163] A. Troccoli and P. K. Allen. Relighting acquired models of outdoor scenes. In *Proceedings of 3DIM*, pages 245–252, 2005.

[164] Z. Tu, X. Chen, A. L. Yuille, and S.-C. Zhu. Image parsing: Unifying segmentation, detection, and recognition. *International Journal of Computer Vision*, 63(2):113–140, 2005.

[165] D. Turmukhambetov, N. D. Campbell, S. J. Prince, and J. Kautz. Modeling object appearance using context-conditioned component analysis. In *Computer Vision and Pattern Recognition (CVPR)*, pages 4156–4164, 2015.

[166] J. R. Tyler, D. M. Wilkinson, and B. A. Huberman. Email as spectroscopy: Automated discovery of community structure within organizations. In *Communities and technologies*, pages 81–96. Kluwer, B.V., 2003.

[167] A. Velten, T. Willwacher, O. Gupta, A. Veeraraghavan, M. G. Bawendi, and R. Raskar. Recovering three-dimensional shape around a corner using ultra-fast time-of-flight imaging. *Nature Communications*, 3:745, 2012.

[168] S. Vicente and L. Agapito. Balloon shapes: reconstructing and deforming objects with volume from images. In *International Conference on 3D Vision*, pages 223–230, 2013.

[169] S. Vicente, J. Carreira, L. Agapito, and J. Batista. Reconstructing PASCAL VOC. In *Computer Vision and Pattern Recognition (CVPR)*, pages 41–48, 2014.

[170] S. Vicente, C. Rother, and V. Kolmogorov. Object cosegmentation. In *Computer Vision and Pattern Recognition (CVPR)*, pages 2217–2224, 2011.

[171] X. Wang and E. Grimson. Spatial Latent Dirichlet Allocation. In *Neural Information Processing (NIPS)*, pages 1577–1584, 2007.

[172] O. Wasenmüller, M. Meyer, and D. Stricker. CoRBS: Comprehensive RGB-D benchmark for SLAM using Kinect v2. In *Winter Conference on Applications of Computer Vision (WACV)*, 2016.

[173] T. Whelan, S. Leuteneggera, R. F. Salas-Moreno, B. Glocker, and A. J. Davison. ElasticFusion: Dense SLAM without a pose graph. In *Robotics Science and Systems (RSS)*, 2015.

[174] K. Wu and M. D. Levine. Recovering parametric geons from multiview range data. In *Computer Vision and Pattern Recognition (CVPR)*, pages 159–166, 1994.

[175] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015.

[176] J. Xiao, A. Owens, and A. Torralba. SUN3D: A database of big spaces reconstructed using SfM and object labels. In *International Conference on Computer Vision (ICCV)*, pages 1625–1632, 2013.

[177] C. Xiong, D. Johnson, R. Xu, and J. J. Corso. Random Forests for metric learning with implicit pairwise position dependence. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 958–966, 2012.

[178] B. Zheng, Y. Zhaoy, J. C. Yuy, K. Ikeuchi, and S.-C. Zhuy. Beyond point clouds: Scene understanding by reasoning geometry and physics. In *Computer Vision and Pattern Recognition (CVPR)*, pages 3127–3134, 2013.

[179] Q. Zheng, A. Sharf, G. Wan, Y. Li, N. J. Mitra, D. Cohen-Or, and B. Chen. Non-local scan consolidation for 3D urban scenes. *ACM Transactions on Graphics*, 29(4):94, 2010.

[180] Q.-Y. Zhou and V. Koltun. Dense scene reconstruction with points of interest. *ACM Transactions on Graphics*, 32(4):112, 2013.

[181] J.-Y. Zhu, J. Wu, Y. Wei, E. Chang, and Z. Tu. Unsupervised object class discovery via saliency-guided multiple class learning. In *Computer Vision and Pattern Recognition (CVPR)*, pages 3218–3225, 2012.

[182] X. Zhu and A. B. Goldberg. *Introduction to Semi-Supervised Learning*. Morgan & Claypool, 2009.